Automated Analysis of Security Protocols with Movement in UPPAAL

Andrew Cook

Supervisor: Prof. Luca Viganò

A thesis presented for the degree of

Doctor of Philosophy

Department of Informatics

King's College London

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text.

Andrew Cook

2022

Abstract

Standard approaches to the formal and automated analysis of security protocols typically do not consider explicitly the fact that certain protocols require the (honest and dishonest) agents to move to specific locations in order to send or receive particular messages. In this thesis, I formalise a formal and automated approach in which security protocols with explicit agent movement are analysed in the presence of a Dolev-Yao attacker. I define movements as timed processes in an applied pi-calculus that allows me to build traces that show how the location and the movements of the agents and of the attacker significantly affect not only the execution of a protocol but also the execution of the possible attacks. I have automated my approach by using the UPPAAL tool. As proof-of-concept, I show my calculus and its UPPAAL implementation in action by analysing several security properties of two concrete case studies.

Contents

1 Introduction **10**

1.1 Overview 11

1.1.1 Research Aims 12

1.1.2 Thesis Outline 14

1.2 Related Work 15

2 Background and Motivations **21**

2.1 Protocols and Security 22

2.1.1 Encryption and Decryption 26

2.1.2 Integrity and Cryptographic Hashing 27

2.1.3 Authentication and Digital Signatures 28

2.1.4 Trade-Offs and the Myth of Perfect Security 29

2.1.5 Security Protocol Notations and Conventions 30

2.1.6 A Note on the Safety of Cryptography 33

2.1.7 Attacking the NSPK Protocol 34

2.2 The Dolev-Yao Attacker Model 37

2.2.1 Extending the Dolev-Yao Attacker Model 38

2.2.2 A Dolev-Yao Attacker Extended With Movement . . . 39
2.3 A Survey of Security Analysis Tools . . . 42

3 A Calculus of Mobile Protocol Agents 46
3.1 Overview . . . 47
3.2 Messages, Time, and The Environment . . . 47
3.2.1 Time . . . 48
3.2.2 The Environment . . . 49
3.3 The Nodes and the Environment . . . 50
3.3.1 Well-formed Environments . . . 50
3.3.2 Node Processes . . . 51
3.4 The Hubs . . . 53
3.5 Message Bridge Rules . . . 56
3.6 The Attacker . . . 57
3.7 Conclusions . . . 57

4 Analysis of Security Protocols with Movement In UPPAAL 61
4.1 Overview and Related Work . . . 62
4.2 Security Protocol Modelling in UPPAAL . . . 63
4.2.1 Message Passing . . . 63
4.2.2 Movement . . . 64
4.2.3 Cryptography . . . 65
4.2.4 Embedding The Attacker Model in UPPAAL . . . 67
4.2.5 A Note On Environments versus Scenarios . . . 71
4.3 Embedding the Calculus . . . 72

4.3.1 Message Passing, Manipulation, and Knowledge 72
4.3.2 Cryptography 75
4.3.3 Models of the Initiator, Responder, and Attacker . . . 76
4.3.4 Analysis and Verification 88
4.4 EnvGen: Another Layer of Automation 97
4.4.1 Motivations 97
4.4.2 Using EnvGen and Obtaining Results 100
4.5 Conclusions 110

5 Case Studies **112**

5.1 Case Study:
The New-Bank-Card Protocol 113
5.1.1 The Protocol in The Calculus 113
5.1.2 UPPAAL Implementation 116
5.1.3 Instantiating the Scenario 121
5.1.4 Analysis 123
5.1.5 Conclusions 129
5.2 Case Study: The SegCom protocol 135
5.2.1 The Protocol in Our Calculus 136
5.2.2 UPPAAL Implementation 137
5.2.3 Instantiating the Scenarios 143
5.2.4 Analysis 144
5.2.5 Conclusions 149

6 Conclusions and Future Work **153**

6.1 Conclusions 154
6.2 Future Work 155

List of Figures

2.1 The Two Generals Problem 23

2.2 Needham Schroeder Public Key Protocol 32

2.3 Lowe's Attack on the Needham Schroeder Public Key Protocol 35

2.4 Needham-Schroeder-Lowe Public Key Protocol 36

3.1 Message Generation and Analysis Rules 49

3.2 Example of a (Well-Formed) Environment 51

3.3 Message-Passing Rules for Nodes 53

3.4 Movement Rules for Hubs 55

3.5 Message-Passing Rules for Hubs 55

3.6 Message-Passing Bridge Rules 56

4.1 Example of Sending a Message 64

4.2 Example of Receiving a Message 64

4.3 General Schema of an Attacker Brain 69

4.4 Cryptography Code for NSPK with Movement in UPPAAL . . 75

4.5 Nonce Creation for NSPK with Movement 77

4.6 The Model of the NSPK Initiator 77

4.7 A Choice of Keys for the Initiator 78
4.8 The UPPAAL Model of the NSPK Responder 79
4.9 The UPPAAL Model of the Attacker 80
4.10 An Example Scenario of NSPK with Movement 82
4.11 A Trace of Alice in The Calculus 82
4.12 A Trace of Bob_1 in The Calculus 83
4.13 A Trace of Bob_2 in The Calculus 83
4.14 A Trace of Bob_3 in The Calculus 84
4.15 UPPAAL Model of the Environment 84
4.16 The Instantiation of the Movement Part of Alice 85
4.17 The Instantiation of Message Passing Part of Alice 86
4.18 The Instantiation of Bob 87
4.19 The Instantiation of the Movement Part of the Attacker . . . 88
4.20 The Instantiation of the Message Passing Part of the Attacker 89
4.21 A Trace of a Possible Attack on Security Property 4.3.2 92
4.22 A Trace of a Possible Attack on Security Property 4.3.3 95
4.23 A trace of an attack on Security Property 4.3.4 96
4.24 A Comparison of an Environment and its Equivalent Planar Graph 99
4.25 The User Interface of EnvGen 100
4.26 Selecting a Pre-existing Model in EnvGen 101
4.27 Instantiating Alice 102
4.28 Instantiating Bob 103
4.29 Instantiating Attacker 104

4.30 Selecting a Pre-Existing Query File 105
4.31 The Generated NSPK With Movement Files With Unique Environments 106
4.32 An EnvGen-Created Environment in Which All Nodes Overlap 107
4.33 EnvGen Analysis In Action 108
4.34 Example Attack on Security Property 4.3.4 in an EnvGen-Created Environment 109

5.1 The New-Bank-Card Protocol 115
5.2 The Environment for the Scenario of The New-Bank-Card Protocol 117
5.3 UPPAAL Model of the Initiator in the New-Bank-Card Protocol 118
5.4 UPPAAL Model of the Responder in the New-Bank-Card Protocol 119
5.5 UPPAAL Model of the Attacker Brain in the New-Bank-Card Protocol 120
5.6 Instantiation of the Scenario for Analysis in UPPAAL 122
5.7 The Instantiation of the Movement Part of the Customer . . . 123
5.8 The Instantiation of the Message Passing Part of the Customer 123
5.9 The Instantiation of the Bank 124
5.10 The Instantiation of the Bank Employee 125
5.11 The Instantiation of the Attacker Brain 126
5.12 The Instantiation of the Movement Part of the Attacker . . . 127
5.13 Attack on Security Property 5.1.1 132
5.14 A Trace of a Possible Attack on Security Property 5.1.4 133
5.15 A Trace of a Possible Attack on Security Property 5.1.5 134

5.16 A Scenario of the SegCom Protocol 137
5.17 Signing and Verification Code for the SegCom Protocol in UPPAAL 138
5.18 An Example MITM Attack on SegCom with Unsigned Certificates 139
5.19 UPPAAL Model of the Initiator in the SegCom Protocol . . . 140
5.20 UPPAAL Model of the Responder in the SegCom Protocol . . 141
5.21 A Choice of Keys for The Initiator 141
5.22 A Choice of Keys for The Responder 142
5.23 UPPAAL Model of the Attacker in the SegCom Protocol . . . 142
5.24 The Environment for the Scenario of the SegCom Protocol . . 143
5.25 Instantiation of the Scenario for Analysis in UPPAAL 144
5.26 A Choice of Keys for the RSU 144
5.27 A Choice of Keys for the Vehicle 145
5.28 The Instantiated RSU 146
5.29 The Instantiation of the Movement Part of the Vehicle 147
5.30 The Instantiation of the Message Passing Part of the Vehicle . 147
5.31 The Instantiation of the Movement Part of the Attacker . . . 148
5.32 The Instantiation of the Message Passing Part of the Attacker 149
5.33 Trace of a Possible Attack on Security Property 5.2.1 151
5.34 Attack on Security Property 5.2.4 152

List of Tables

3.1 Overview of Movement Rules 54

4.1 Temporal Properties in UPPAAL 88

Chapter 1

Introduction

1.1 Overview

There is a commonly spoken mantra among security professionals that the only truly secure computer is one that is off, unplugged, and buried underground. But the truth is that increasingly complex computer systems play more of a role in our lives than ever before, so securing them becomes an ongoing task. Moore's law dictates that the number of transistors that can be fit onto an integrated circuit on a microchip doubles every two years [36], so the speed and complexity of our computing power shows no signs of slowing down.

With the growth of computing in our everyday lives, principals with malicious intent have wasted no time in keeping up with finding new ways to attack these systems for an array of benefits. Even the term "hacking" has advanced beyond its humble beginnings as a term used by the MIT Tech Model Railway Club for a shared passion for the intricacies of model railways, and now applies to acts of war carried out by nation states. The most striking example of this is the infamous Stuxnet Worm [24], a malicious computer worm that targeted control systems in Iran's nuclear program.

But these systems have not exclusively grown in size in the literal sense since the early days of computing, they have also shrunk - even becoming part of us as humans. Medical devices such as pacemakers have been fitted with wireless capabilities in order for a patient to track their health in real time on a smart device. But these are not necessarily safe from potential attacks either, and indeed Vice President Dick Cheney had the wireless capabilities in his pacemaker specifically disabled so that potential attacks on his life were thwarted in this regard [1].

While security professionals have largely kept the pace with attackers when it comes to securing computing systems, there is never a point where we can stop. Security is an undecidable problem as there are always more actions an attacker can take; not to mention that there is no such thing as perfect security (see § 2.1.4). But what can be done, is to automate the search for potential attacks on computer systems in order to find a sequence of actions that an attacker can take that could be missed by a researcher with a more classical pen-and-paper approach. Automating the approach also has clear benefits with regards to the "explainability" of security, the majority of tools which search for attacks on security protocols will provide a *trace* showing exactly what the attacker is doing and the ways in which they are compromising specific security properties. There are tools that can do such a thing already (see § 2.3) but for this thesis, I consider a different approach that arises from the observation that some security systems involve principals that *move*. Movement is a relatively new metric when it comes to security, largely brought about by the rise of the Internet of Things (IoT). It is obvious that our world is increasingly interconnected, and the simplest of our everyday devices have potential for internet connectivity and communication in cyberspace.

1.1.1 Research Aims

In short, the problem to be addressed is the fact that there are some protocols whose security is dependent on the fact that principals move to certain locations. Not only this, but some protocols can not be executed unless the principals move. The research aims of my thesis are to develop a proof-of-concept

technique for modelling and verifying security protocols with movement. This thesis will show my approach to the problem and demonstrates the efficacy of the approach through two case studies. The case studies are a realistic protocol developed for the thesis between a hypothetical bank card holder and their bank, and a primary authentication phase of a real protocol in literature known as SegCom [43].

My contributions to achieving this are first an extended applied pi-calculus I developed which captures movements of principals in formally defined environments. My extended applied pi-calculus features rules for movement, message passing, and includes bridge rules which link the explicit actions to the implicit actions of each principal. A preliminary version of the calculus appeared in [16], the motivation for which arose from the idea of modelling communicating drones in an environment of base stations. Although the calculus and definition of the environment is different, there is certainly an overlap in motivation.

I next embed the rules of my calculus into the modelling, validation, and verification tool UPPAAL, which I have chosen for its ease of capturing movement (see § 1.2 for more). I also develop and embed an attacker for use in the verification of these security protocols with movement, which contains novel contributions but is inspired by work carried out in [26] [18].

Lastly, the aim of the thesis is to demonstrate the power of the approach I have taken to the problem of security protocols with movement. I model two security protocols in UPPAAL, analysing various security properties related to authentication for both, listing my findings.

1.1.2 Thesis Outline

This thesis first addresses the need to model movement of communicating principals in § 2 and motivates the modified applied pi-calculus to capture movement formally. The calculus is detailed in § 3.

My new calculus rules are embedded into the UPPAAL tool in § 4. As mentioned before I choose UPPAAL primarily for its ease of embedding the applied pi-calculus rules for movement, but also I demonstrate UPPAAL's broader capabilities as a security protocol analysis tool through the use of an example, namely the Needham-Schroeder Public Key Protocol (NSPK). I show how UPPAAL can be used to verify security properties on a security protocol in a scenario of NSPK involving movement.

In § 4.4 I extend UPPAAL in my own unique way to accommodate and broaden the analysis of security protocols under every environmental condition with respect to my definition of an environment in § 3. This extension I call "EnvGen" and exists as a stand-alone application written in Java.

In § 5 I show my approach in action, analysing authentication and agreement properties of two security protocols and listing my results.

1.2 Related Work

This is not the first time that location and movement have been considered in protocol analysis, so let me explain why my approach is different from what has been done so far. *Mobile Ad-Hoc Networks (MANETs)* that use cryptography are a hot research area (see, e.g., [25]). However, in this paradigm, executing security protocols via *routing* is often the preferred tactic than via physical movement and is often more practical. While location and geographical conditions are important to analysing MANET performance [4], in routing protocol attacks such as "black hole" [31], movement of the agents is not considered in the attack trace. Some attacker models in the spirit of the Dolev-Yao model have been developed for MANETs (e.g., [17]), but, to the best of my knowledge, there does not exist a standard, full-fledged, moving Dolev-Yao attacker for MANETs.

Distance-bounding protocols are other cryptographic protocols that securely establish an upper bound on the physical distance between the participating agents [10]. These are particularly useful for proximity-dependent services such as contactless payments or authorised entry to a space in the form of scanning an ID card. In order to symbolically analyse such protocols, a number of security protocol analysis approaches and tools have been extended to consider, in particular, timestamps and the location of agents (see, e.g., [19,33,41]). While there are obviously tight connections with my work, the attacker movement that I consider in § 3 and § 4 is quite different from that usually considered for distance-bounding protocols.

With regards to an applied pi-calculus specifically to model movement,

works such as [12] capture the movement of defined boundaries of computation known as *ambients* between administrative domains. The goal of [12] was to model how a mobile program might gain authorisation to enter or exit certain domains of operation, such as local area networks or working environments. In their own words, Cardelli and Gordon aimed to discover easy and reliable techniques to "transfer a piece of the desktop environment (for example, a forgotten open document along with its editor) to the laptop over a phone line." This work led to the development of Internet applications by treating the Internet as a "globally distributed computing system" [38], or to declarative programming languages such as CLAIM for multi-agent systems [23]. While the ambient calculus is enriched with locations to describe the distributed nature of mobile computation, my notion of location is different. In ambients, a location is defined as an administrative domain within which computation can take place. In my calculus in § 3, a location represents a physical finite range over which computation and communication can happen. Works such as [11] enriched the ambient calculus with notions of security to express permission for an ambient to cross into another administrative domain. My notion of security comes from a protocol specification that exists between (mobile) agents that can communicate when they are in the same location.

Other works have used UPPAAL for security protocol analysis, most notably [18, 26]. In [18], Corin et al. carry out an analysis of protocols that involve timing aspects. The author's motivation for using UPPAAL is both for their "engineering" and for their analysis. UPPAAL allows clocks to be defined over models to monitor the passage of time as the system evolves,

meaning that the tool is a logical choice for engineering protocols "that are aware of timing aspects". These clocks can also be reasoned about with respect to the security properties using UPPAAL, allowing an analysis of timing related attacks for the chosen protocols. My work does not concern timing attacks, and therefore clocks are not utilised. However, my calculus of mobile agents (§ 3) does model a notion of discrete time. Corin et al. include a Dolev-Yao-style intruder (see § 2.2) in their analysis, but there are many message manipulation rules that they do not include. More specifically, the intruder can only assume an identity by way of choosing a public key, encrypt or decrypt a value, generate a nonce, or save a data variable. However, the results of the paper are not for demonstrating the power of the intruder, but more the results they obtain from their analysis and how the protocols can be improved to be more secure. Corin et al. also implement cryptography by way of a "cryptographic device", but I instead opt for the cryptography implementation in [26].

In [26], Koltuksuz et al. also use UPPAAL for its ability for the user to construct timed automata models. Likewise, the tool is used for the analysis of security properties on protocols which are "known to be time sensitive". The attacker model in this paper is much more powerful, allowing for a more complex analysis of all the messages in the protocol as well as more sophisticated message construction. The results of this paper are an analysis on protocols such as the Neumann-Stubblebine protocol with various security properties such as correspondence, agreements, reachability properties, and importantly, timing-related properties, such as not receiving certain messages before they

were sent and so on.

As I will explain in more detail in § 4.2.4, my UPPAAL model of the attacker is based on the attacker "brain" featured in [26] combined with elements of [18], with some improvements and modifications of my own. The main novelty of my implementation lies in the definition of an attacker model that consists of two parts: a novel attacker agent that formalises movement, and that interfaces with an attacker brain to send and receive cryptographic messages. For example, a message can only be received by the attacker brain if the attacker agent has moved to the required location. The second novelty is that my attacker can act as a principal of communication itself (i.e. can initiate a run of protocol at will), rather than simply as a medium through which all communication passes (i.e. having full control of the network by *being* the network). My attacker can assume the role of a node or hub (see § 3) depending on the scenario considered, leading to the possibility of discovering new attacks either with movement or without. More modifications and improvements of my own can be found in § 4.2.4.

The two papers [18] [26] have demonstrated the use of UPPAAL as a tool for building timed automata to simulate time-sensitive protocol executions and analysing timing-related security properties. In this thesis however, I show that UPPAAL can be used to analyse protocols in the context of moving agents, and it comes as naturally as a timing-based analysis.

However, UPPAAL is not designed specifically to analyse security protocols. Therefore, designing an attacker model that captures the Dolev-Yao rules with certain carefully considered restrictions in such a way that the analysis

terminates is an important aspect to the thesis. As seen in the analysis of my case studies in § 5.1 and § 5.2 though, UPPAAL either terminates with the result that the security property in question is verified to hold with respect to the scenario and attacker model, or UPPAAL terminates and returns an attack trace violating the security property in question. Hence, in this thesis I will provide a small set of authentication, agreement, and location verification properties for each case study to produce a variety of results that demonstrate the approach. After all, the main goal of this thesis is to show how my approach enables the analysis of security protocols with movement, rather than to carry out a thorough analysis of a specific protocol, regardless of whether it is a realistic commercial protocol like the New-bank-card protocol § 5.1 or a real protocol like the SegCom protocol § 5.2.

Other tools have been designed specifically for security protocol analysis. For instance, many large-scale protocols have been analysed with Tamarin [35], e.g., [7] and [15] to name just a couple. In [15], Chothia and Ryan use Tamarin to verify security properties of an IEEE standard key management protocol. Basin et al.'s main contribution in [7] is a formalisation of the 5G network standard and a "full, systematic security evaluation" of one of the authenticated key exchange protocols from the family of 5G protocols using Tamarin. Even though Tamarin is specifically designed for security protocol analysis, the authors still note that a "substantial, real world protocol" such as 5G AKA is challenging to model.

Still, Tamarin obviously fares much better at security protocol analysis than UPPAAL as it has been designed and further tailored to that purpose.

However, to carry out the analysis that I want to with movement would involve extending the Tamarin tool itself or capturing movement semantics in a fairly non-trivial way. With UPPAAL, instead, capturing movement is much simpler and more flexible for my needs. This convenience is the reason why I chose to use UPPAAL in this thesis, as a proof-of-concept, even though there is nothing in my approach that would prevent its use with Tamarin or one of the other more efficient security protocol analysis tools. In fact, I might do so in future work to analyse protocols with movement more complex than those considered here.

Chapter 2

Background and Motivations

2.1 Protocols and Security

There exists a thought experiment first proposed is 1975 by Akkoyunlu et al. that aimed to show that two or more entities communicating through unreliable means could not possibly hope to co-ordinate an action [2]. The thought experiment became known as the "two generals problem", and was important in the development of coordination of computer networks. It describes a problem which is analogous to communicating over a medium that is potentially compromised by a malicious principal referred to henceforth as an *attacker*. Although this thought experiment is not directly related to computer security per se, it provides a solid example to demonstrate how security researchers treat computer networks and the Internet.

The problem is as follows: two generals leading armies are aiming to attack a city. However, both armies need to launch their attacks simultaneously to be guaranteed victory. The only way the two generals can communicate is by sending messengers through a valley occupied by the city's defenders. This means the generals can not be guaranteed that their messages (perhaps suggesting a time to attack, or indeed a confirmation that the message was received) were received by the other general. What the generals might do, is send many messengers in the hopes that at least one reaches the other general, in which case they can be reasonably confident that their message was received. Ultimately, though, neither general can be guaranteed whether the last message they sent had been received or not.

Taking this example of the two generals further, guaranteeing that a message is delivered is only part of the problem. The messages carried by the

Figure 2.1: The Two Generals Problem

messengers might be readable by the city's guards, who could then appropriately prepare for an attack. The messages could be changed in transit by a traitorous messenger, for example. The generals can protect themselves against potential attacks from attackers by employing the use of a security protocol.

A *protocol* is simply a set of instructions that determine how one should act in a given scenario. A security protocol is a protocol that invokes security properties on messages, using techniques such as *cryptography*, to ensure that communicating parties can be as confident as reasonably possible that messages they exchange are delivered "securely". However, the mere presence of cryptography is not enough to guarantee that security properties hold. Furthermore, a security property does not "prevent" an attacker from compromising messages, a security property either holds or does not for a security protocol due to cryptographic techniques employed. That said, the cryptographic principles underpinning the many different security properties remain the same across almost all security protocols.

Cryptography is defined by Rivest as "communication in the presence of adversaries" [39]. It is the act itself of applying cryptographic algorithms to a message to ensure that the message is bound by certain security properties.

Different algorithms bind different security properties to messages, each of which limits how an attacker (such as the city guards) might be able to exploit the information contained in the message. Here are some of the most common security properties common to many security protocols.

- **Confidentiality** is the property that data is not disclosed to unauthorized parties. In the case of the Two Generals problem, if the plan of attack was written down the sender might choose to *encrypt* the messages using a shared key, for example. The second General would then *decrypt* the messages using the same shared key. A subcategory of confidentiality is known as **privacy**, and asserts that individuals have full control over which information about them can be disclosed, collected, or stored.

- **Integrity** is the guarantee that data has not been tampered with by unauthorized parties. This would be akin to the message carried by a messenger being tampered with by an attacker.

- **Availability** is the guarantee that data and services are always easily accessible by authorized parties. Capturing messengers to prevent Generals communicating is an obvious violation of this security property.

- **Authentication** is the term ascribed to two different types of security properties. Firstly, *data origin authentication* describes that the receiver of data which claims to be created by a certain principal is confident that the data was created by that principal, (i.e. "The General wrote this message"). The second type of authentication, *entity authentication*, is

that one principal can be confident that the claimed identity of another principal is correct (i.e "I am confident I am talking to the General and not someone else").

- **Anonymity** is quite the opposite of authentication. Firstly, *untraceability* describes the property that data created by a communicating principal cannot be traced back to the principal that created it. Anonymity can also be associated with the identity of a principal, such as their name or IP address.

- **Non-Repudiation** is the property ascribed to an action such that the principal that carried it out cannot later deny that they carried out that action. A simple example of this would be if the General signed each message. Only the signature holder can create their own signature so it is irrefutable who signed the document, and the General cannot later claim that they never sent a signed message.

Leaving the analogy of the two generals now, all of the above security properties are also used in the context of computer security, the topic of this thesis. Computers are very efficient at applying cryptographic algorithms to accomplish different security properties on messages exchanged over a computer network. In Sections 2.1.1 and 2.1.2 I introduce some cryptography which is used to accomplish some of the security properties that will become relevant.

2.1.1 Encryption and Decryption

There are a number of cryptographic algorithms to accomplish confidentiality, but together they are known as *encryption algorithms*. Encryption algorithms (referred to henceforth as E) always convert a plain-text, m, to a cipher-text with the use of a key, k_1.

$$E(m, k_1) = c$$

The cipher-text can be decrypted using a *decryption algorithm*, D, with a key k_2.

$$D(c, k_2) = m$$

Encryption algorithms can be either *symmetric* or *asymmetric*. Symmetric means that $k_1 = k_2$ or knowing one key allows easy derivation of the other. Common symmetric encryption algorithms include the Advanced Encryption Standard [20]. In symmetric encryption generally, two or more communicating parties establish together a singular shared key, which they use to encrypt all messages. In asymmetric encryption, the communicating principals each create a *public* and *private* key pair, instead of just a singular shared key. A receiver's public key can be used by anyone, and is used to send encrypted messages to them, which they decrypt using their private key and read. The security of asymmetric encryption stems from the fact that the algorithms used to create the key pairs render it computationally infeasible to deduce the private key from a public key. For asymmetric enryption, the standard is typically RSA

(Rivest-Shamir-Adelman) [40].

The cipher-text is sent to a receiver. I take a symbolic instead of a probabilistic approach to cryptography in this thesis. Therefore the attacker, who is presumed to be monitoring all communication (more in Section 2.2), cannot read any information from the ciphertext; it is incomprehensible to anyone who cannot *decrypt* the ciphertext with the correct key. This is how confidentiality is accomplished. The messages are confidential, so long as the sender and receivers both keep either their shared symmetric keys or their private keys secret.

2.1.2 Integrity and Cryptographic Hashing

Integrity is the guarantee that data has not been tampered with by unauthorized principals. One way to accomplish this, hash functions are used. A hash function is a function $h(x) = y$ which takes an input and produces an output known as a message digest or simply "hash". The significance of hash functions is that they have the property of being one-way, meaning that it is computationally infeasible to find an x given a y such that $h(x) = y$. The hash y is also always of a fixed and shorter length meaning the hash function h provides compression. A *cryptographic* hash function is a hash function parameretised with a key, i.e. $h(x, k) = y$. Cryptographic hash functions have the additional property that it is computationally infeasible to compute y given an x without knowing the key k. Cryptographic hash functions typically also have one of two additional properties. Second pre-image resistance is the property that given an input x_1, it is computationally infeasible to find

another input x_2 such that $h(x_1) = h(x_2)$. Collision resistance is the property that it is computationally infeasible to find two inputs x_1 and x_2 such that $h(x_1) = h(x_2)$.

Accomplishing data integrity between a sender and a receiver typically involves using a family of cryptographic hash functions known as Message Authentication Codes (or MACs). A message authentication code is a family of hash functions h_k parameterised by a secret key k. A sender and a receiver share a key k, and h_k must have the property of being *computationally resistant* (i.e. given zero or more MAC pairs $(m_i, h_k(m_i))$, it is infeasible to compute $(m, h_k(m))$ for any new input $m \neq m_i$). The sender computes the digest of a message m, i.e. $h_k(m)$. The sender then sends to the receiver $(m, h_k(m))$. If we assume that an attacker will always try to tamper with the message m and the MAC $h_k(m)$, to produce m' and $h_k(m)'$. The receiver can verify the integrity of m' that they received by recomputing $h_k(m')$ with the received m'. If $h_k(m') = h_k(m)'$, then the message was not tampered with while in transit. The security of MACs comes from the fact that the attacker cannot compute $h_k(m')$, so long as the sender and receiver keep the keys secret.

2.1.3 Authentication and Digital Signatures

Digital signatures offer proof of data-origin and non-repudiation. When one signs a document, it is impossible to later deny having signed that document. The same can be accomplished for messages sent between principals in a computer security context, and it is accomplished using public key cryptography.

In general, a digital signature scheme for a principal A is defined first by a

set of messages $\mathcal{M}$ which can be signed, and a set $\mathcal{S}$ of signatures. Next, the principal A is equipped with two functions S_A and V_A, where $S_A : \mathcal{M} \rightarrow \mathcal{S}$ is a signing function which A can use to sign messages, and $V_A : \mathcal{M} \times \mathcal{S} \rightarrow \{true, false\}$ is a verification scheme for A's signature and is publicly known. To sign a message, A creates a signature for $m \in \mathcal{M}$ by computing $S_A(m)$ and sending the pair (m, s). A receiver B for instance, can verify A's signature by computing $u = V_A(m, s)$ and accepts the signature if $u = true$. For digital signature schemes to be protected from attacks, it should be difficult for any other principal other than A to compute $V_A(m, s)$ for any $m \in \mathcal{M}$ and $s \in \mathcal{S}$ where $V_A(m, s) = true$.

In practice, digital signatures can be accomplished using RSA cryptography (see 2.1.1). RSA decryption with a private key acts as a signing function, while using the encryption function with a principal's public key acts as the verification.

2.1.4 Trade-Offs and the Myth of Perfect Security

An important point is that these cryptographic techniques are easily combined to improve the security of messages. For example, sender A and receiver B can agree to use message authentication codes. This gives the messages integrity and thus no attacker who may have compromised the communication could tamper with the messages. Sender A and receiver B can also then decide to encrypt the messages that are sent, thereby sending an encrypted message m, and a MAC. This would mean that the messages have combined confidentiality and integrity thereby strengthening their security.

However, security is not simply a matter of combining all security properties together. Not only are some of the security properties the inverse of each other (i.e. authentication and anonymity), but in some cases the impracticality of employing certain cryptography outweighs the necessity for it. Cryptography, particularly public key cryptography, is very computationally intensive, making this less practical. For lightweight and efficient confidential communication one might choose to use symmetric encryption instead. This decision though comes with its own drawbacks of course, namely the problem of *key distribution* i.e. safely distributing keys to honest parties without dishonest parties getting a hold of them, which is solved by public key cryptography.

There is no such thing as perfect security. Securing communication is done with security protocols and cryptography, but what is important is to make considered decisions that reflect the *scenario* in which the cryptography is used.

2.1.5 Security Protocol Notations and Conventions

In the case of the Two Generals Problem, the two generals have a common goal, which is to not only exchange information but to do so in such a way that their messages are secure. Between them exists what is referred to (particularly in the context of computer security) as a *channel* of communication. Depending on the cryptography that is used, security properties will hold for these channels. Such security properties can be defined with respect to a sender A and a receiver B as the following.

- An *insecure* channel between A and B is one that provides no security

guarantees.

- A *confidential* channel guarantees that if a sender A sends a message to receiver B, A can rely on that fact that only B can receive the message.
- An *authentic* channel guarantees that if A sends a message to B, B can rely on that fact that A has sent the message and meant it for B.
- A *secure* channel is one that is both confidential and authentic.

A channel is merely abstract and refers only to the means of communication that exist between one or more communicating parties. Traditionally, the series of messages that form a security protocol that are sent across a channel can be expressed symbolically to convey a "run". A run of a security protocol can be expressed in a conventional notation known as "Alice and Bob notation" [40], sometimes referred to simply as security protocol notation.

Cryptograpic principles and security properties are visible in Alice and Bob notation. The following are standard notation to represent cryptography in a security protocol.

- **Asymmetric Encryption:** $\{m\}_{K_X}$ - a message m encrypted with the public key of some communicating principal X. The corresponding private key for principal X is written as K_X^{-1}
- **Symmetric Encryption:** $\{\!|m|\!\}_{K_{XY}}$ - a message m encrypted with a symmetric key shared between communicating parties X and Y.
- **Concatenation:** (m_1, m_2) - two messages m_1 and m_2 concatenated together. For readability, the parentheses will be omitted and nested

concatenations such as $((m_1, m_2), m_3)$ shall be written shorthand as m_1, m_2, m_3.

- **Signing:** $\{m\}_{K_X^{-1}}$ - a message m signed with the private key of a communicating principal X.

Figure 2.2 depicts an example of a simple security protocol known as the Needham-Schroeder Public Key Protocol [37] using Alice and Bob notation.

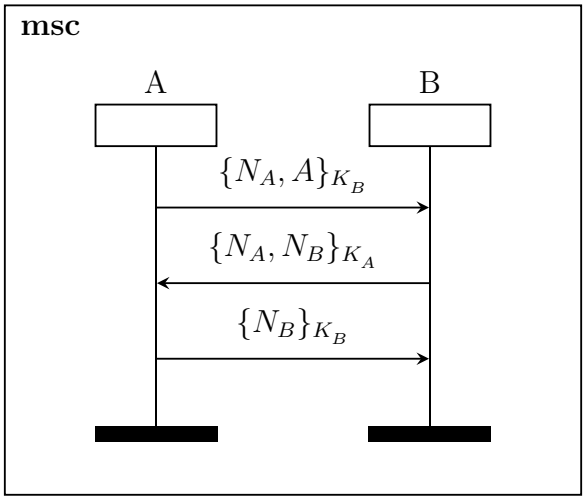

Figure 2.2: Needham Schroeder Public Key Protocol

The Needham-Schroeder Public Key protocol is an elementary security protocol that attempts to achieve mutual authentication, but is actually vulnerable to attacks (see Section 2.1.7).

The first message sent by Alice to Bob, denoted in Figure 2.2 as "A" and "B", contains a freshly created *nonce* N_A concatenated with Alice's name, "A". A nonce is a freshly generated random number that is intended to be used only once and has no visible association to Alice. The concatenated N_A and A is encrypted with Bob's public key (as it is intended to be read only by

Bob). Bob replies by sending back Alice's nonce concatenated with a nonce created freshly by Bob, N_B, all encrypted with Alice's public key K_A. Finally Alice sends just Bob's nonce encrypted with Bob's public key to Bob.

Bearing in mind that security protocols are intended for use over computer networks, these messages are typically sent as bit streams across the channel connecting Alice and Bob. The purpose of the nonces is to ensure the freshness of the messages being exchanged. Because the nonces are randomly generated, there is no way that Bob could have a pre-prepared response to Alice's initial message; he must decrypt the message and use the nonce Alice created in his reply. In order to do that, Bob's private key must be indeed secret and known only to him.

2.1.6 A Note on the Safety of Cryptography

Standards of security analysis as we know them were introduced upon the advent of public-key encryption in the mid 1970s. Before this, symmetric key cryptographic schemes such as the Data-Encryption Standard (DES), developed by IBM in the early 1970s, were known already to be prone to brute-force key search attacks. Due to the computing power at the time this was fairly impractical, but as technology improved computers could easily begin decrypting encrypted messages by simply searching and trying all the 2^{56} possible 56 bit DES keys in a matter of hours. This meant that symmetric keys agreed upon by two or more principals of communication only had a certain safe lifespan before they had to be renewed for security purposes.

The most widely-used form of public key encryption is the Rivest-Shamir-

Adleman (RSA) cryptosystem [40] mentioned previously in Section 2.1. This cryptosystem's function relies on large prime numbers, and its security is dependent on the fact that currently there is no known algorithm to easily factorise large numbers into two prime numbers. The keys themselves are much larger than symmetric keys, meaning that while these keys could still technically be recovered by brute-force key search attacks, this method would be infeasible. While brute-forcing an RSA key and factorising large numbers into primes is currently known to be computationally infeasible, this does not mean other possible attacks are not present which do not involve knowing the key. Section 2.1.7 gives an example of such an attack on a protocol known as the Needham-Schroeder Public Key Protocol.

2.1.7 Attacking the NSPK Protocol

Often when security protocols are designed, they are found later to be vulnerable to attacks by an *attacker*. Attackers are typically thought of as either being active or passive [42]. A passive attacker is "an unauthorized attacker which eavesdrops on the communication between two parties in order to steal information stored in a system by wiretapping or similar means." An active attacker can "transmit data to all the parties, or block the data transmission," in addition to the capabilities of a passive attacker.

The Needham-Schroeder Public Key protocol is inherently flawed, as was found by Gavin Lowe in 1995 [30]. In Lowe's paper, he provides an example of an attack on NSPK where "an intruder can impersonate an principal A during a run of the protocol, to trick another principal B into thinking that

he really is talking to A." The Alice and Bob notation of the attack is given in Figure 2.3.

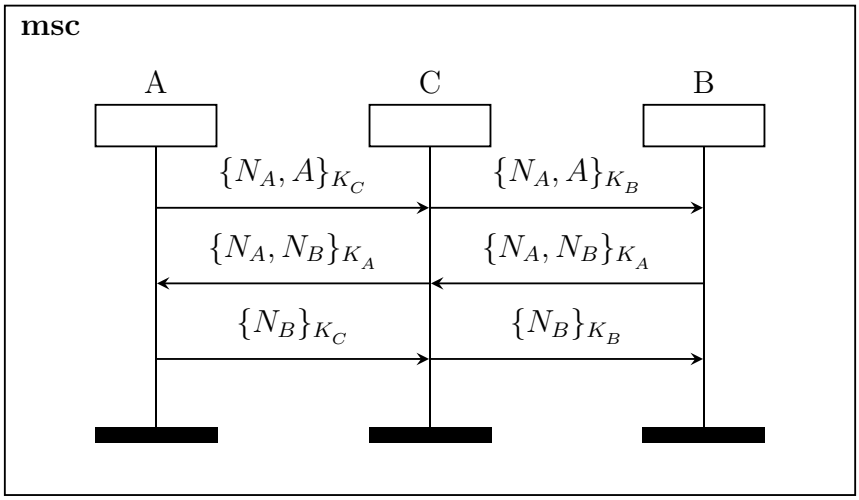

Figure 2.3: Lowe's Attack on the Needham Schroeder Public Key Protocol

In this attack, the attacker is named "Charlie" and acts as a *man-in-the-middle* in order to attack this protocol. Alice initiates the protocol with Charlie as normal, and has no reason to believe that Charlie will act in bad faith to any other principal. Charlie receives the first message from Alice and decrypts it. Charlie then re-encrypts the content of this message with Bob's public key and sends the message to Bob, who believes that he is talking to Alice due to the inclusion of her name in the message. Charlie receives a message encrypted with Alice's public key which he cannot decrypt. However, this is no matter, he simply sends that message on to Alice. Alice receives the nonce she created and a newly created nonce N_B. As per the protocol, she returns this nonce, but to Charlie. The end result is that Bob believes he is talking to Alice, but is in fact talking to Charlie. The problem here is

that Alice has no way of knowing the origin of N_B. To her, this is simply a nonce like any other. This disrupts the ultimate goal of the protocol, which is mutual authentication.

This attack is prevented by the addition of Bob's name in the reply to Alice, giving the Needham-Schroeder-Lowe protocol.

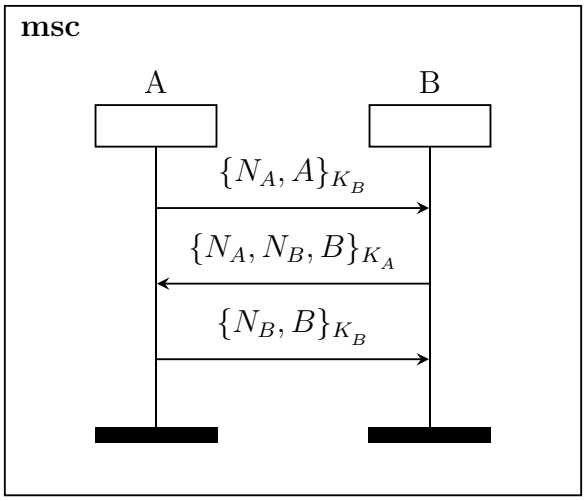

Figure 2.4: Needham-Schroeder-Lowe Public Key Protocol

With the inclusion of Bob's name in the second message, when Alice receives and decrypts this message, she sees Bob's name included. If Charlie were to attempt the same attack, she would see that the message she received from Charlie would have Bob's name suspiciously included, and would abort the protocol, since it is clear that Charlie is attempting a man-in-the-middle attack with her and Bob. Even with this fix though, the protocol is still vulnerable to other attacks, namely a type-flaw attack that relies on principal's confusion between nonces, names, and keys [34].

Designing security protocols which are resistant to attacks is a difficult

problem. That said, attacker *models* have been developed over the years and have since been used extensively in protocol security analysis to uncover novel attacks. Perhaps the most pervasive model, the Dolev-Yao attacker model (see Section 2.2) has been embedded into state-the-art protocol analysis tools in order to perform automatic analysis of protocols to verify if security properties hold in them or not (see Section 2.3).

2.2 The Dolev-Yao Attacker Model

In order to analyse and verify security protocols, some approaches have been developed in literature over the years that have become standard. If one considers the most powerful attacker [13] to control the network, then the cryptographic measures taken to protect messages from this attacker will also naturally protect the messages from weaker attackers with fewer capabilities. This kind of attacker is known as the *Dolev-Yao Attacker Model* [21].

In the Dolev-Yao attacker model, the attacker can send a message to any principal of communication, intercept all messages, build a message, read a message, modify messages, and so on. However, the only capability that the attacker is assumed not to have is breaking encryption (i.e. the attacker does not have knowledge of how to factorise large numbers into primes in the case of RSA, since there is no known mathematical procedure to easily do this.) This is a widely used standard attacker model when analysing security protocols [5]. The rules of the Dolev-Yao attacker are detailed in Section 3.2.

Since its advent, the Dolev-Yao attacker has played a part in many security

protocol analyses. An advantage of being able to formally define the rules and capabilities of the attacker is that it can be modelled and can be embedded into automatic protocol analysis and verification tools (see Section 2.3).

However, the Dolev-Yao attacker has also been enhanced and extended in various ways as technology and computer systems have become more complex and interconnected to be better suited to the analysis.

2.2.1 Extending the Dolev-Yao Attacker Model

While the Dolev-Yao attacker model is considered to have full control over a network, there are some networks where this is not enough to describe the possible actions of the attacker. This is mostly due to the advancement of the Internet of Things (IoT), and the increasing network connectivity of everyday devices and infrastructure. For instance, autonomous vehicles typically contain an entire network of sensors that continuously measure the speed of the vehicle based on the current incline, speed limits, and the distance to vehicles in front and behind. An attacker might be able to spoof sensor readings on these senors in any number of ways.

It is important to account for extra actions that do not strictly align with the notion of a passive or active attacker in the conventional sense of sending and receiving messages on a network. In a work by Lanotte, Merro, Munteanu, Viganò [29] the Dolev-Yao attacker model was extended to account for the capacity of the attacker to interfere with sensor data in cyber-physical systems (CPS). CPS are computer systems with "the ability to interact with, and expand the capabilities of, the physical world through computation, com-

munication, and control [3]." Lanotte et al. define a cyber-physical system as "real-time control systems, a hierarchy of sensors, actuators, and control components (that) are connected to control stations."

The Dolev-Yao attacker has been extended in other ways, for instance in [8], the authors propose an extension of the Dolev-Yao model to include a notion of computational time. In [32], the authors propose a modified version of the Dolev-Yao attacker which is more appropriate for analysing *socio-technical systems* and human-computer interaction. The authors' approach was to abstract away technical aspects of the Dolev-Yao model to be better suited to common human-computer interactions such as printers and Bluetooth pairing. Even more recently, in the age of quantum computing and quantum security protocols, there have been works that bring the Dolev-Yao attacker into the quantum realm, allowing for security protocol analysis on *probabilistic* protocol logic, as opposed to a strictly symbolic one à la Alice and Bob notation.

2.2.2 A Dolev-Yao Attacker Extended With Movement

While it is true that the Dolev-Yao attacker is assumed to have complete control over the network, questions arise when the network itself becomes more dynamic; perhaps spread across vast geographical areas, with principals of communication joining and leaving the network periodically. Not only this, but the attacker itself might might need to *move* to be in the right place at the right time to be able to launch an attack on a security protocol.

As such, the dynamic nature of the network itself might be intrinsic to the execution of the protocol, and dishonest principals in this network might need

to account for this. This is thus the basis of this thesis. While in the current model, the Dolev-Yao attacker is presumed to have complete control over the network; all messages pass through the attacker. But the current model says nothing about the *environment* in which the attacker conducts attacks. In order for a dishonest principal to have full control of a network, they might need to plan, time, and execute movements effectively in an environment. They might need to meet a principal at a certain destination at a certain point in the protocol execution, otherwise the attack fails.

While one could easily imagine a situation in NSPK for instance where Alice must move in order to complete the protocol with Bob, (and indeed this is done in 4.2) it is important to consider intuitive every-day protocols involving movement to better motivate the planned extension to the Dolev-Yao attacker model.

Motivation: The New-Bank-Card Protocol

In order to motivate this extension to the Dolev-Yao model, take for example a simple scenario where your debit or credit card has expired and you must acquire a new one. This is essentially a protocol between you and your bank. The protocol is initiated by the bank, when the card expires. Once this happens, you must go to the bank, show your details to the teller, and order a new card. Not only is this a protocol, but it is a security protocol since typically one has to authenticate themselves through some means to the bank to complete the protocol.

In several countries worldwide, in order to request a new card, a customer

needs to take an existing active card to their local bank branch and enter their personal identification number (PIN) when requested by the teller, and a new card will be automatically sent to the billing address on file. The new card only becomes the currently active card once the customer uses it at an ATM (or cash point) and enters the card's PIN number. Note that here we are not interested in talking about a real protocol, but are content with a realistic one for illustrative purposes. A description of the customer's actions in such a protocol could then be:

- Leave the current address.
- Go to the bank.
- Present card, enter PIN, request new card.
- Leave bank.
- Go to billing address (possibly different from the current address).
- Receive new card from bank.
- Leave billing address.
- Go to ATM.
- Present new card, enter PIN.

This is an example of a protocol that simply *does not work* unless the human moves to the required locations in order to send and receive the corresponding messages. Note that some of the messages exchanged in this protocol

are “standard” digital messages or can be encoded in that way (such as the customer entering the PIN and pushing a button to request a new card at the ATM), whereas other ones are “analogic” messages delivered by a physical postal service (such as the bank sending an envelope containing the new card to the customer’s billing address). There is no need to distinguish between these types of messages in a cryptographic context, although it would be interesting to consider explicitly how to formalise them in a way that is true to their real nature.

If one wishes to consider movement and location to be a fundamental aspect of a security protocol, then one must consider how an attack can be influenced by its inclusion. This thesis presents an approach to this problem with a discussion of the results obtained from a number of security protocol analyses (Sections 5.1 and 5.2).

2.3 A Survey of Security Analysis Tools

There are a plethora of tools that have been designed for security protocol analysis. This subsection provides an overview of some of the tools that have been used in research. All of the tools are created for the purposes of automatically verifying security protocols or analysing if properties are held by protocols, but they all differ in their modus-operandi.

The Tamarin Prover [35] is a state-of-the-art tool for symbolic protocol analysis and verification. In Tamarin, protocols are modelled using multiset rewriting rules and properties are specified using first-order logic. Tamarin can

perform automatic verification of properties with a protocol and it also supports interactive manual verification. If the tool terminates when analysing a property, it either provides a proof that it holds for the protocol, or it returns a counter-example. In the case of the tool finding a counter-example, a dependency graph is provided which is a "partially ordered set of rule instances that represent a set of executions that violate the property" [6]. As mentioned in § 1.2 Tamarin was used recently to uncover security vulnerabilities in protocols within the 5G network [7]. In this paper, Tamarin was used to carry out a formal analysis of a model of the 5G AKA protocol with respects to the security goals listed in the 3GPP standard.

ProVerif is another widely-used tool suited to the analysis and verification of properties of security protocols [9]. ProVerif models protocols and the attacker as Horn theories, and using ProVerif one can specify a cryptographic protocol and associated security properties using ProVerif's own input language, based on the applied-pi calculus. ProVerif allows for an unbounded number of protocol sessions that may run concurrently and without putting a bound on the size of messages an attacker can generate, however, work that uses ProVerif [28] have cited difficulty in modelling "algebraic operators such as the exclusive OR (XOR) and Diffie-Hellman exponentiation." ProVerif much like Tamarin has been used to uncover vulnerabilites in deployed technology. For example, in [14], a hardware chip known as a the Trusted Platform Module was found to have vulnerabilities that the authors discovered by modelling authorisation protocols within it. More recently, ProVerif was used to analyse properties on authentication protocols associated with 5G enabled device-to-

device communication [22].

Maude-NPA is a protocol analysis tool that is the next generation of the NRL Protocol Analyser. Maude-NPA is a model checker that operates on unbounded sessions, and operates on backwards searching. That is, the tool searches backwards from an insecure state to determine whether or not it is reachable from the initial state. Maude-NPA is also a symbolic analysis tool. If the tool terminates (which is may not) without finding an attack, this provides a formal proof that the protocol is secure with respect to the specified security property.

UPPAAL (https://uppaal.org) is a tool for the modelling, validation, and verification of inter-communicating systems. UPPAAL allows one to build machines that are graphically represented and can be simulated in real time, and UPPAAL allows verification on instantiated labelled transition systems with respect with respect to user-defined properties in UPPAAL's "query language". If UPPAAL verification terminates then the property in question either holds, or UPPAAL produces a diagrammatic *trace* showing a counter-example to the property. Unlike in Tamarin or ProVerif, where security protocols are specified using a formal input language, UPPAAL allows the user to build detailed labelled transition systems through a GUI.

UPPAAL does not have a pre-existing model of an attacker controlled network, but it does allow different labelled transition systems to communicate via global *channels* and exchange information via global variables (more in § 4.2). UPPAAL has been used in this way before [26] [18] to analyse and verify properties of security protocols, mostly because of UPPAAL's real-number

valued clock variables which can be used for finding timing based attacks on protocols. Perhaps the most striking difference about UPPAAL is that it is *not* a symbolic model checker. That is, all data involved in the model is typed.

However, this thesis demonstrates the power of using UPPAAL for general purpose protocol analysis and verification of security properties (see § 4.2). Not only this, but because UPPAAL's save-files of models are stored in the XML format and UPPAAL developers provide an API, this thesis shows how this can be taken advantage of for powerful and thorough analysis of protocols involving *movement* in formally defined environments (see § 3 and § 4.4).

Before this new extended attacker can be embedded into the UPPAAL tool though, in Section 3 I first extend the capabilites of the attacker on paper, defining first what is an *environment* and the ways in which an attacker can move around an environment.

Chapter 3

A Calculus of Mobile Protocol Agents

3.1 Overview

In this chapter, I formalise my approach to model and reason about security protocols with movement. I give an applied pi-calculus that formalises messages, time, communication, and an environment that principals move in to exchange messages.

3.2 Messages, Time, and The Environment

My approach is independent of the specific language of messages, but for concreteness I consider the standard operators for: *message pairing* (m_1, m_2), or simply m_1, m_2; *symmetric encryption* $\{\!|m_1|\!\}_{m_2}$; and *asymmetric encryption* $\{m_1\}_{m_2}$. Other operators (e.g., hashing, exponentiation, XOR) can be added easily.

For a set M of messages, I define $\mathcal{DY}(M)$ to be the smallest set closed under the standard *generation (G)* and *analysis (A)* Dolev-Yao-style rules given in Figure 3.1; rules for other operators can be added as necessary. These rules model how messages can be constructed/deconstructed but say nothing about sending/receiving messages, which will be handled by other rules. The rule G_{axiom} states that if a message m is in the message knowledge set M, then it must be in the closure of the Dolev-Yao knowledge $\mathcal{DY}(M)$. G_{pair} states that if two messages m_1 and m_2 are in the Dolev-Yao knowledge $\mathcal{DY}(M)$, then the constructed pair (m_1, m_2) is in the Dolev-Yao closure $\mathcal{DY}(M)$. The two rules G_{scrypt} and A_{scrypt} both relate to symmetric encryption and decryption. The first rule G_{scrypt} states that if m_1 and m_2 are in the Dolev-Yao knowledge

$\mathcal{DY}(M)$, then the symmetric encryption of m_1 with m_2 is too. Conversely, A_{scrypt} states that if a symmetrically encrypted message $\{\!|m_1|\!\}_{m_2}$ is in the Dolev-Yao knowledge $\mathcal{DY}(M)$ and so is m_2, then m_1 is in the Dolev-Yao knowledge as it can be easily obtained by decryption. Lastly, G_{crypt}, A_{crypt}, and A^{inv}_{crypt} relate to asymmetric cryptography. G_{crypt} states that if m_1 and m_2 are in the Dolev-Yao knowledge, then the encryption $\{m_1\}_{m_2}$ is in the Dolev-Yao knowledge. The rules A_{crypt} and A^{inv}_{crypt} both relate to decryption in the asymmetric context; firstly A_{crypt} states that if m_1 encrypted with m_2 is in the Dolev-Yao knowledge and the inverse of key m_2 is in the Dolev-Yao knowledge, then m_1 is in the Dolev-Yao knowledge as it can obtained easily by decryption. A^{inv}_{crypt} states the same as A_{crypt} except that m_2 and the inverse of m_2 are swapped. This is a consequence of the fact that for some asymmetric encryption schemes such as RSA, the public and private keys are pairs which can be swapped for signing / verification and so on.

3.2.1 Time

Following [16], a *global, linear clock* gives the calculus and the protocols a notion of time. The clock is a global variable t that annotates each process in the calculus, and is incremented by the function $inc(t)$. Sending or receiving messages takes no time, but movement of the protocol principals *does* make time pass (more on this later).

$$\frac{m \in M}{m \in \mathcal{DY}(M)}\ G_{axiom} \qquad \frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{(m_1, m_2) \in \mathcal{DY}(M)}\ G_{pair}$$

$$\frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{\{\!|m_1|\!\}_{m_2} \in \mathcal{DY}(M)}\ G_{scrypt} \qquad \frac{m_1 \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{\{m_1\}_{m_2} \in \mathcal{DY}(M)}\ G_{crypt}$$

$$\frac{(m_1, m_2) \in \mathcal{DY}(M)}{m_i \in \mathcal{DY}(M)}\ A_{pair_i} \qquad \frac{\{\!|m_1|\!\}_{m_2} \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{m_1 \in \mathcal{DY}(M)}\ A_{scrypt}$$

$$\frac{\{m_1\}_{m_2} \in \mathcal{DY}(M) \quad inv(m_2) \in \mathcal{DY}(M)}{m_1 \in \mathcal{DY}(M)}\ A_{crypt}$$

$$\frac{\{m_1\}_{inv(m_2)} \in \mathcal{DY}(M) \quad m_2 \in \mathcal{DY}(M)}{m_1 \in \mathcal{DY}(M)}\ A^{inv}_{crypt}$$

Figure 3.1: Message Generation and Analysis Rules

3.2.2 The Environment

The *environment* (as in [16]) is modelled as a two-dimensional affine plane of *nodes*, which are static principals with a finite circular range of communication that exchange messages with mobile, point-like entities that are called *hubs*.

In the following, I first refine the environment from [16] in terms of the positions of the nodes relative to one another (§ 3.3). I then formalise how nodes can exchange messages with hubs (§ 3.3.2) and how hubs can interact with the environment by moving around and exchanging messages with nodes (§ 3.4). I also introduce some bridge rules (§ 3.5) and extend the Dolev-Yao attacker with movement (§ 3.6).

3.3 The Nodes and the Environment

Nodes in my environment are circles of uniform radius, which are either far apart or overlapping with each other. Each node has a *name* that represents, e.g., the name of a Wi-Fi receiver or of a road-side unit in a vehicular ad-hoc network. The radius models the "range of communication" of the node.

Definition 3.3.1 (The Environment) *An* environment *Env is a pair* (ϕ, O)*, where* ϕ *is a set of* node names *and* O *is a relation describing the* overlaps *between nodes.*

In order to establish an upper-bound on the size of *Env*, I impose some sensible restrictions on a node's location: a node cannot "contain" another node (i.e., a smaller node inside a larger node), and only two nodes may *mutually* overlap (i.e., three or more nodes may not all mutually overlap, although a node may overlap with more than one other node as shown in Figure 3.2, where $\phi = \{x, y, u, v\}$ and $O = \{(y, u), (u, v)\}$). Nodes are also restricted from being positioned directly on top of another, and two nodes may not be just touching at their circumferences.

3.3.1 Well-formed Environments

Environments can be thought of as planar graphs (more in § 4.4), where the graph nodes represent *our* nodes and edges represent the overlaps. Planar graphs are graphs whose edges only intersect at the nodes (i.e. the edges do not cross each other in the plane). A simple observation is that the complete graph of five nodes K_5 is non-planar. Kuratowski's Theorem [27] states that

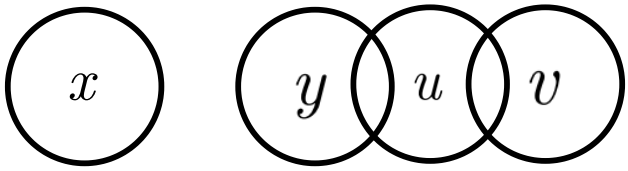

Figure 3.2: Example of a (Well-Formed) Environment

a graph is planar if and only if it contains no subdivisions of K_5 (or $K_{3,3}$, a complete bipartite graph on six vertices). From this, it follows that there is no such environment where $O = \phi \times \phi$ for $|\phi| > 4$. Therefore I will not consider these forbidden environments from now on. Environments that are not forbidden are henceforth planar, as Kuratowski's theorem states.

Definition 3.3.2 (Well-formed Environment) *An environment Env is* well-formed *if, for* $O \subseteq \phi \times \phi$ *and for all* $(n_1, n_2) \in O$, $n_1 \neq n_2$ *and* $(n_2, n_1) \notin O$ *and* $|O| < 5$ *and Env is planar.*

From now on, I shall only consider well-formed environments, like, e.g., the environment in Figure 3.2. Take for instance, though, the environment $(\{n_1\}, \{n_1, n_1\})$. This is not well-formed, since O defines n_1 to overlap with itself.

3.3.2 Node Processes

Following [16], a node's *trace* is described by the behaviour of a process n. In particular, n_i is a named node process, where $0 \leq i < |\phi|$ and $|\phi|$ is the size of ϕ. For a finite set of nodes ϕ, there is a finite set of *process names* $\{\mathsf{n}_0, \mathsf{n}_1, \ldots\}$, where $|\{\mathsf{n}_0, \mathsf{n}_1, \ldots\}| = |\phi|$ and $0 \leq i, j \leq |\{\mathsf{n}_0, \mathsf{n}_1, \ldots\}|$, described by the grammar

$$
\begin{array}{rll}
\mathsf{n}_i, \mathsf{n}_j ::= & U(m).\mathsf{n}_i & \text{input} \\
| & \overline{U}\langle V \rangle.\mathsf{n}_i & \text{output} \\
| & \mathsf{n}_i \parallel \mathsf{n}_j & \text{composition} \\
| & \tau.\mathsf{n}_i & \text{internal} \\
| & 0 & \text{halt}
\end{array}
$$

The processes are quite standard to applied pi-calculus: $U(m).\mathsf{n}_i$ is the action of receiving a variable m on a channel U and evolving into process n_i; $\overline{U}\langle V \rangle.\mathsf{n}_i$ is the action of sending the known message construct V over a channel U before evolving into n_i; $\mathsf{n}_i \parallel \mathsf{n}_j$ represents multiple processes occurring simultaneously; $\tau.\mathsf{n}_i$ represents the standard internal transition; 0 halts the process.

Each node (in fact, each protocol principal) x possesses a *message knowledge set* K_x^t that represents what messages x knows at global time t based on the messages that it knows initially, has received or freshly generated. Nodes are able to send/receive messages at will, on their default channel name. I model this by means of the rules $Read_{\mathsf{n}}$, $Write_{\mathsf{n}}$ and $LinkWrite_{\mathsf{n}}$ in Figure 3.3.

Reading a message is a transition that a node should be able to do at any time, so the rule $Read_{\mathsf{n}}$ has no premises; at any time instant, t, a node is able to receive a message m on its default channel x. The rules for writing, on the other hand, have preconditions that reason about the node's knowledge, and the conclusions are transitions that the node process can perform: $Write_{\mathsf{n}}$ requires that the message to be sent must be part of the node's current knowledge at the time t, whereas *LinkWrite* allows a node x that overlaps with another node y to directly send a message to y.

$$\frac{}{x(m).\mathsf{n}_i^t \xrightarrow{x(m)} \mathsf{n}_i^t}\ Read_{\mathsf{n}} \qquad \frac{m \in \mathcal{DY}(K_x^t)}{\overline{x}\langle m\rangle.\mathsf{n}_i^t \xrightarrow{\overline{x}\langle m\rangle} \mathsf{n}_i^t}\ Write_{\mathsf{n}}$$

$$\frac{m \in \mathcal{DY}(K_x^t) \quad (x,y) \in O}{\overline{y}\langle m\rangle.\mathsf{n}_i^t \xrightarrow{\overline{y}\langle m\rangle} \mathsf{n}_i^t}\ LinkWrite_{\mathsf{n}}$$

Figure 3.3: Message-Passing Rules for Nodes

3.4 The Hubs

Hubs are modelled as point-like entities in the plane that can move in and out of the ranges of nodes, as in [16]. If a hub is in range of a node, then it is able to exchange messages with the node. If a hub is out of range of the node, then it must move in order to exchange messages with the node. If a hub is in the overlapping region of two nodes, then it is able to exchange messages with both nodes without the need to move.

A hub's *trace* is described by the behaviour of a process h. In particular, h_i is a named process of a hub, where $0 \leq i \leq |\mathcal{H}|$ with $\mathcal{H}$ the set of hub processes. The hub processes follow the same applied-pi calculus grammar as the one for node processes in § 3.3.2.

In addition to its message knowledge set, each hub h_i possesses a *location set* $E_{\mathsf{h}_i}^t$ that describes its location at time t: if a hub enters a node, the channel name of this node is added to its location set; upon leaving, the channel name of this node is removed from its location set.

In order to model movement and describe the trace of a hub in the environment, I introduce the rules that are given in Figure 3.4 and summarised in Table 3.1. These rules describe different possible movements that a hub can

Table 3.1: Overview of Movement Rules

Rule	**Description**
A_{single}	Into a single node.
D_{single}	Out of a single node.
A_{double}	Into the overlapping region of two nodes.
D_{double}	Out of the overlapping region of two nodes.
A_{cross}	Between two nodes that overlap.
OA_{single}	When in one node that overlaps with another, moving into the overlap.
OD_{single}	When in an an overlap of two nodes, moving to be just inside one.

make: they define how a single hub's location set is updated when the hub moves into and out of nodes.

These rules are of the form $\frac{J_1}{C}$ or $\frac{J_1 \quad J_2}{C}$, where J_1 is a judgement on the location set of a hub $E^t_{\mathsf{h}_i}$ and J_2 is a judgement about the structure of Env, and the conclusion C is an updated location set $E^{inc(t)}_{\mathsf{h}_i}$.

A_{single} and D_{single}, for example, formalise how a hub h_i's location set is updated when it moves into range and out of range of a single node in the environment. Note that a single node can be far away from all other nodes *or* overlapping with other nodes. The premises of A_{single} show that the hub is in empty space, i.e., its location set $E^t_{\mathsf{h}_i}$ is empty, and that a node x exists. The conclusion, after the movement, is that the hub's new location set $E^{inc(t)}_{\mathsf{h}_i}$ is a set containing just x. The meaning of the other rules is as described in Table 3.1.

Note that these rules provide an operational semantics for movement in

$$\frac{E^t_{\mathsf{h}_i} = \{\} \quad x \in \phi}{E^{inc(t)}_{\mathsf{h}_i} = \{x\}}\ A_{single} \qquad \frac{E^t_{\mathsf{h}_i} = \{x\}}{E^{inc(t)}_{\mathsf{h}_i} = \{\}}\ D_{single}$$

$$\frac{E^t_{\mathsf{h}_i} = \{\} \quad (x,y) \in O}{E^{inc(t)}_{\mathsf{h}_i} = \{x,y\}}\ A_{double} \qquad \frac{E^t_{\mathsf{h}_i} = \{x,y\}}{E^{inc(t)}_{\mathsf{h}_i} = \{\}}\ D_{double}$$

$$\frac{E^t_{\mathsf{h}_i} = \{x\} \quad (x,y) \in O}{E^{inc(t)}_{\mathsf{h}_i} = \{x,y\}}\ OA_{single} \qquad \frac{E^t_{\mathsf{h}_i} = \{x,y\} \quad (x,y) \in O}{E^{inc(t)}_{\mathsf{h}_i} = \{x\}}\ OD_{single}$$

Figure 3.4: Movement Rules for Hubs

$$\frac{x \in E^t_{\mathsf{h}_i}}{x(m).\mathsf{h}^t_i \xrightarrow{x(m)} \mathsf{h}^t_i}\ Read_{\mathsf{h}} \qquad \frac{m \in \mathcal{DY}(K^t_{\mathsf{h}_i}) \quad x \in E^t_{\mathsf{h}_i}}{\overline{x}\langle m\rangle.\mathsf{h}^t_i \xrightarrow{\overline{x}\langle m\rangle} \mathsf{h}^t_i}\ Write_{\mathsf{h}}$$

Figure 3.5: Message-Passing Rules for Hubs

well-formed environments, as they state that at any time each hub can be in only one of three locations: in an empty space outside of any node (I will also refer to this empty space as "no man's land"), inside a node, or inside the overlap of two nodes. It is impossible for a hub to be in a location different from these three, e.g., inside two non-overlapping nodes or inside three overlapping nodes at the same time. Moreover, hubs cannot jump or "teleport" as they can move from a node x to another node y only if the two nodes overlap, and must otherwise first exit x and then enter y.[1]

The rules $Read_{\mathsf{h}}$ and $Write_{\mathsf{h}}$ in Figure 3.5 allow a hub to send and receive messages. A hub is always able to receive a message from a node as long as they are within range of that node. So, the only premise for the rule $Read_{\mathsf{h}}$ is that a

[1]This operational semantics of movement suffices for my purposes in this thesis, but a more fine grained operational semantics is possible, along with a denotational one. I will consider both in future work.

$$\frac{x(m).\mathsf{p}^t \xrightarrow{x(m)} \mathsf{p}^t}{\{m\} \cup K^t_{\mathsf{p}}}\ \textit{ReadMsg} \qquad \frac{\nu(m).\mathsf{p}^t \xrightarrow{\nu(m)} \mathsf{p}^t}{\{m\} \cup K^t_{\mathsf{p}}}\ \textit{FreshMsg}$$

Figure 3.6: Message-Passing Bridge Rules

hub's location set includes a node x. If so, much like the nodes, $\overline{x}\langle m\rangle.\mathsf{h}^t_i$ evolves into process h^t_i when it receives a variable m on a node's default channel.

For rule $\textit{Write}_{\mathsf{h}}$, the action $\overline{x}\langle m\rangle.\mathsf{h}^t_i$ can evolve into h^t_i if x is in h^t_i's location set and m is in the Dolev-Yao closure of the message knowledge set of h^t_i (i.e., the hub has to be able to construct m from the rules in Figure 3.1).

3.5 Message Bridge Rules

When I build a trace in the calculus, there are some state transitions that are implicit, particularly the message construction rules. However, the *bridge rules* in Figure 3.6 allow the processes to traverse from the world of the implicit to the explicit.

The rules *ReadMsg* and *FreshMsg* bridge between *any* node/hub/attacker process denoted generically as p and the Dolev-Yao knowledge. *ReadMsg* takes the premise that a process p has read a message m, and the process can then add the message m to its message knowledge set K^t_{p}. *FreshMsg* takes as premise the standard creation of a fresh m using the ν operator, and concludes by adding m to p's message knowledge set K^t_{p}.

3.6 The Attacker

The attacker a is a dishonest principal that can assume the role of a node or a hub at will. In short, it is a Dolev-Yao attacker that I have extended with movement: it is capable of creating and sending messages by way of the Dolev-Yao message rules, as well as receiving messages either directly on its channel name or by eavesdropping messages sent on another channel only if it has moved within range.

I denote the process of the attacker as a who inherits all the same capabilities of honest nodes and hubs, as in § 3.3.2. The attacker is likewise in possession of a message knowledge set K^t_{a} and location set E^t_{a}.

3.7 Conclusions

In this chapter I have created an extended applied pi-calculus, which is able to model the communication and movement of principals involved in a security protocol by building a trace composed of sequential and parallel inferences of the rules. Although applied pi-calculus has been extended in other ways for different needs, my particular calculus and its rules are suited directly to modelling the kinds of movements that a principal can take within the bounds of my formally defined well-formed environments.

My model of an attacker in this context is one that can assume the role of a hub or node at will. A standard Dolev-Yao attacker is assumed to have full control over a network, therefore in this thesis an all-powerful attacker that has full control over a network should be able to have full control over

the environment also. Of course, the movement of the attacker is what I am interested in, so it would be a rather uninteresting result to assume that the attacker occupies and controls *all* possible physical space in an environment. At the classical communication level, it is safe to assume the attacker has full control over messages sent and received by principals. At the physical level, I believe it is safe to assume the attacker *could be* any principal.

Although all possible movements in a two-dimensional affine plane are accounted for in Figure 3.4, there is an implicit notion of a *normal* trace of movement. Both a dishonest and honest hub would never decide to take a "long" route into an overlap of two nodes (i.e moving in to one of the two overlapping nodes before the overlapping region itself) since this simply wastes time and potentially other vital resources a mobile principal would possess. Thus, one can consider a normal trace of a security protocol with movement to be a trace in which no hubs make unnecessary movements. In the subsequent chapters, I only consider normal traces when providing exemplary calculus traces for scenarios of security protocols in question. Furthermore, when the process becomes automated, the UPPAAL tool will only provide normalised traces as diagnostic traces for attacks in the various case studies, as will be seen in § 4 and § 5.

The rules themselves by no means propose a final state of affairs for any and all types of physical movement a communicating principal might use. In future work, environments may become infinitely more complex, and in my research I encountered some of these complexities myself. For example, nodes which are allowed to wholly contain other nodes call for a need for

much more complex rules. Environments can become potentially geometrically arbitrarily small and even recursive (i.e. fractals), and the definition of the environment and the rules would need to change to account for this. My definition of the environment is not designed to accurately model a specific real world environment (or indeed a specific geometry), but I believe they are sufficient enough to capture the notion that a security protocol might only be executable by a principal moving in some specific way among physically separated principals. Conversely, the notion of overlapping is more analogous to the fact that a hub *does not* need to move in order to communicate with more than one principal of communication.

As will be seen in the case studies in § 5, nodes may also act simply as "meeting points" for hubs to communicate with each other. The node essentially acts as the network that allows communication between two hubs by simply forwarding messages on. This follows implicitly from the message passing rules already provided so I do not include specific rules for hub-to-hub communication. If a node acts dishonestly even as a meeting point between two hubs, its direct influence on the communication is already accounted for in the Dolev-Yao rules and in the embedding of the calculus in § 4.

The following chapter showcases an in-depth look at the modus-operandi of UPPAAL's most important features for this thesis, and showcases how the calculus in this chapter is embedded into the tool. Through this embedding of the rules, I can thus allow an automated search for attacks on security protocols with movement which may or may not provide traces of attacks that resemble those which might be found by using my calculus with a classic pen-and-paper

approach. The advantages of this show not only how UPPAAL can be used as a security protocol analysis tool, but also how UPPAAL is particularly suited for capturing movement as extra factor in a protocol specification.

Chapter 4

Analysis of Security Protocols with Movement In UPPAAL

4.1 Overview and Related Work

UPPAAL (https://uppaal.org) is a tool for the modelling, validation, and verification of inter-communicating systems. UPPAAL allows one to model inter-communicating systems by building labelled transition diagrams. The validation aspect of UPPAAL is in reference to the interactive nature of the tool: one can manually explore the state space of their system and validate behaviours. The verification functionality of UPPAAL allows one to check properties of the system using the "query language" (see § 4.3.4 of the tool, and optionally produce diagnostic traces showing that either a provided property holds or a counterexample trace violating this property.

This section provides an in depth look at the various ways in which UPPAAL is used, and offers a documentation of the ways in which I have used it.

4.2 Security Protocol Modelling in UPPAAL

The following techniques are essential to capturing the capabilities of the principals, and describe the methods in which the communication rules, cryptography, and movement abilities given by our calculus in § 3 are embedded into UPPAAL.

4.2.1 Message Passing

First, the reading and writing rules from Figure 3.3 are embedded in UPPAAL (the rule *FreshMsg* will be embedded in § 4.3.2).

Messages, and components of messages, are conventionally [18, 26] defined as integers of a fixed length of bits in UPPAAL. As such, in order for one principal to send a message to another principal, the former must write an integer to a global variable named "message", which the other principal may then read. As UPPAAL calls for typed models, the way that I type protocol messages is a novel, admittedly minor, contribution; I define integer *ranges* that represent different types of message components in a protocol. Message *components* are typed parts of a message (names, ID's, nonces, etc.) that may be encrypted, signed or hashed. Message components are concatenated through bitwise operations the same way as in [18, 26] (more in § 4.3.1), but in UPPAAL the integer type *int* ranges between -32768 and 32767, making the maximum length of any message 15 bits.

For a time-negligible communication to take place, it is necessary that the sender and receiver *synchronise* on an UPPAAL *channel*. For example, if a

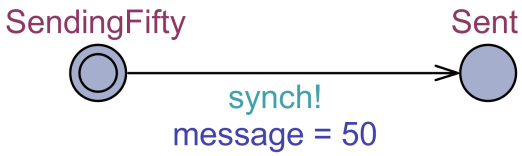

Figure 4.1: Example of Sending a Message

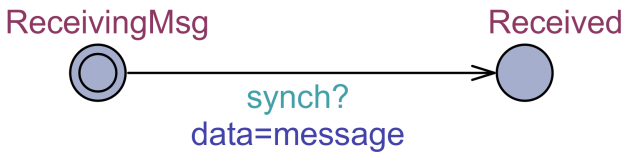

Figure 4.2: Example of Receiving a Message

system A wants to send to a system B the message 50, then A needs to be in a state where it could make a transition labelled with the channel name *synch*! and B needs to be in a state where it can make a transition labelled with the same name *synch*?. When this synchronisation is activated by A, UPPAAL processes this transition first, setting the global variable *message* to the value 50. Then UPPAAL processes B's transition and reads the value of *message* into a local variable for further use. This technique is shown in Figures 4.1 and 4.2.

4.2.2 Movement

Next, the rules from Figure 3.4 are embedded in UPPAAL. Movements are simple to model in UPPAAL (which is indeed part of the reason why I have chosen to use UPPAAL). I create a model that captures the layout of the environment. For all principals which are hubs, the environment is instantiated with boolean variables that monitor that specific hub's location set. These boolean variables are true or false if the hub is in that location or not. As

such, any number of hubs may be instantiated to move around the environment and can freely move in and out of nodes.

Each node in the environment is modelled as a state that a hub can be in should they chose to move there. If a hub is not in any node, it is modelled as being in *no man's land* which can be captured by a single state named *NML*. An overlapping region of two nodes is itself a unique location and therefore UPPAAL state, for instance if two nodes named A and B overlapped, the overlap region is a separate state *AoB* (A overlap B), and so on.

4.2.3 Cryptography

This subsection simply gives an outline of the techniques that will be used for cryptography, but the implementation of these techniques is discussed in detail in § 4.3.2.

As in [26], cryptographic operations in UPPAAL are carried out by means of a pair of arrays that all principals are permitted to read and write to. The arrays allow any number of principals to perform encryption and decryption, sign messages, and verify signatures. Other cryptographic operations such as hashing can be added whenever necessary. To encrypt, a principal calls the *encrypt* function with parameters *param1* and *param2* respectively containing the plaintext and key. The *plain* array stores the plaintext and the *key* array stores the key k in their next free indexes, and returns this index number. This index number is the *ciphertext* c.

To decrypt, a principal supplies a ciphertext c and the key k' to *param1* and *param2* to function *decrypt*. The function returns the plaintext value

stored at index c in *plain* if $key[c] = k'$, returning 0 otherwise.

For symmetric encryption, k and k' will be equivalent if the principal has used the correct key. For asymmetric encryption, a principal supplies a plaintext p and public key pk and a ciphertext is returned. For decryption, the function carries out a simple computation f on the supplied secret (private) key $f(sk')$, and checks if $pk = f(sk')$ at index c. This computation cannot be carried out by principals in order to guess or compute other principals' private keys. If they match, the function returns the plaintext at the index c in the *plain* table.

To sign a message, a principal supplies the plaintext and its secret key in *param1* and *param2* to the *sign* function, respectively. Much like encryption and decryption, the index s of a table of signed messages is the signature itself. Of course, signed messages have a different type than encrypted messages.

To verify a message, a principal supplies the signature s and the sender's public key to the *verify* function, and *true* is returned for a valid signature and *false* otherwise.

The two advantages of embedding cryptography through publicly available functions are: (1) that an arbitrary number of principals can be given cryptographic functionality (similarly to movement), (2) that the underlying mathematics of encryption schemes are abstracted away, which are assumed to be unbreakable by the attacker during analysis and verification regardless.

4.2.4 Embedding The Attacker Model in UPPAAL

As mentioned in § 3.6, the attacker can assume the role of a hub or node at will (i.e. inheriting the capabilities of both). In practice, this amounts to the attacker being either a node (or several nodes), or any number of collaborating hubs *or both*, depending on the scenario being analysed. If the attacker is a hub (or any collaborating number thereof), I instantiate *Attacker Agents*, principals that can move into any of the nodes in the environment. Because I do not restrict them, the attacker agent's movements at the right time in an execution of the protocol could be a key aspect in an attack.

Once an attacker agent has moved into a node, it interfaces with the second part, the *attacker brain*. The attacker brain can send/receive messages through the communication channels that are available to it. In other words, if the attacker agent is present in a node (or is the node itself) where communication is taking place, then all communication passes through the attacker brain who can build and analyse messages according to the Dolev-Yao message generation and analysis rules. Additionally, the attacker brain can send messages to a chosen recipient which is in the same location as one of its attacker agents. All the attacker agents share this same attacker brain. The attacker brain is based on the attacker model given in [26] combined with elements of [18], with some further restrictions, optimisations, and modifications of my own.

As in [18, 26], the state space quickly explodes when the attacker brain is included in the UPPAAL model of the protocol. I must thus carefully restrict the attacker brain in UPPAAL in such a way that the automated analysis of the protocol will terminate. Of course, with any restrictions on a state space,

possible attack traces could be lost, so this begs the question of which attacker brain restrictions I choose and why. I introduce the following four restrictions.

(1) *Typing*: Every message component is properly typed. This includes nonces (fresh data) that are created by any principal. The attacker brain is prevented from creating nonces of a type that would not type check or pattern-match with a receiver. So, like many security protocol analysis tools, I do not consider type-flaws in the searches for attacks. This is a minor consequence of my chosen tool, UPPAAL. It is wasteful to build messages that will be immediately rejected by a receiver who type checks all messages. This is especially important for nonces. It is arguable over whether attacks are lost because of this (but to formally prove this is a further research question).

(2) *Message Length*: The maximum length of a message that the attacker brain can construct for a given protocol is bounded. This restriction is what allows the search for attacks to terminate, as there are now only a finite number of messages that the attacker can *construct*. The "length" of a message is determined by how many message components (e.g., names, ID's, nonces) have been concatenated together. As mentioned in § 4.2.1, for each case study all message components in the UPPAAL models are of a pre-set fixed length of bits, no matter if they are encrypted, signed, hashed, or otherwise. Thus, this bound is determined by the maximum length of a message that pattern matches with a receiver.

(3) *Shallow Cryptography*: The attacker brain is only allowed to perform some cryptographic technique (e.g., en-/decrypting) *each* only once, every time it sends a message. The justification for this restriction is because in the two

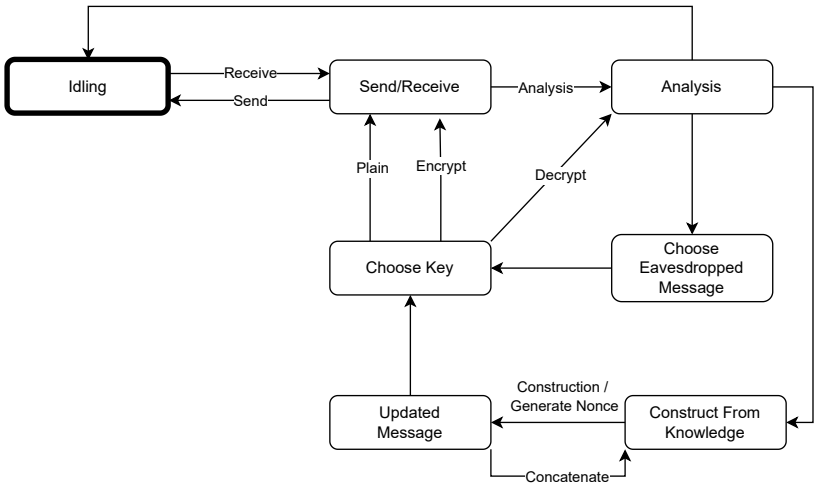

Figure 4.3: General Schema of an Attacker Brain

case studies in § 5.1 and § 5.2, the protocols do not nest encryptions. For other protocols, I can of course lift this restriction at the risk of UPPAAL running out of memory more often before attacks are found.

(4) *Selective Cryptography*: The attacker brain should not be able to use cryptographic techniques that have no relevance to a given protocol. For instance, if a protocol uses only symmetric encryption, then public-key cryptographic capabilities are not available to the attacker brain.

The basic overview of the attacker brain is given in Figure 4.3.

However, for each protocol that I wish to analyse in UPPAAL, the attacker brain will need to be constructed in UPPAAL separately for that protocol, as the data involved vary from protocol to protocol. The schema remains the same. This is not an indictment of the attacker brain model itself or indeed the attacker models from [18, 26], but rather of the UPPAAL tool itself.

The main novelty of my implementation lies in the ability of the attacker

to move, by means of novel attacker agents that all interface with an attacker brain, i.e., the attacker brain can receive a message only if an attacker agent has moved into the required node. My implementation as opposed to [26] has been altered to better suit the specific restrictions of my implementation in order to make my verification and analysis more efficient. This I have done in the following ways:

- Instead of acting purely as an intermediary in principal communication, the attacker themself can initiate a protocol with a principal.
- Choosing a key and carrying out encryption is the very last thing the attacker brain performs before a message is sent.
- Creating a nonce is carried out only during the message construction phase.
- Whenever an encrypted block is received it must either be decrypted and analysed straight away, replayed immediately, or a new message must be constructed.
- "Blocks" (see § 4.3.3) are redefined to be messages to which no more message components can be added. This means they are not message components and therefore not used in message construction, reducing the state space explosion during analysis.
- Once message construction has begun, a full block must be built from the current knowledge before any further action is taken.

- Once a block has been built, it must be sent (encrypted or otherwise). This prevents wasteful actions of the attacker and cyclical construction and discarding of messages in the transition system.

In general, the aim of my modifications is to have the least amount of branching paths that the attacker brain transition system is faced with upon receiving a message, to curb the state space explosion. Cycles in the transition system are avoided wherever possible to give the attacker brain a more systematic approach to follow, and wherever choices must be made by the attacker brain these have been reduced wherever possible in line with the above restrictions.

4.2.5 A Note On Environments versus Scenarios

A well-formed environment is but a part of a whole scenario of a protocol execution. A *scenario* is given by the initial knowledge of each principal and the attacker prior to the protocol execution, which principals are nodes and which are hubs, the starting positions of each principal and the attacker, all in addition to the environment.

By reference to § 3.2.2, the rules of my environments are either that nodes overlap or they do not. In other words, for a finite number of nodes, there are a finite number of possible environments that can be created out of them.

It is possible to generate all possible environments for a given protocol before verifying security properties for each one using UPPAAL (see § 4.4). To firstly demonstrate the efficacy of the tool and the approach, I will consider single environments when describing the initial example in § 4.3 as well as in

the case studies in § 5.1 and § 5.2. However, for verification and analysis I will verify security properties within multiple interesting scenarios to further demonstrate the power of the approach. Each scenario that I study will be described.

4.3 Embedding the Calculus

In this section the NSPK protocol (which is shown in § 2.1.5) is modelled in UPPAAL in a scenario which requires movement among the principals, to show how the rules of the calculus from § 3 are embedded into UPPAAL and how aforementioned modelling techniques are applied for a straightforward analysis and verification of this simple protocol.

4.3.1 Message Passing, Manipulation, and Knowledge

By reference to the protocol itself and the message sequence chart in Figure 2.2, a maximum of two message components can be concatenated together before encryption so I have chosen each message component to be 4 bits. Ciphertexts are themselves message components, therefore an encryption of two message components is indeed one message component. There are three message / message component types to consider; the ID's, the nonces, and the ciphertexts. Since there are 4 bits per message component, I assign the following integer ranges to each type of message component: $1 \leq \text{Ciphertext} \leq 5$, $6 \leq \text{ID} \leq 7$, and $8 \leq \text{Nonces} \leq 15$. The ranges themselves are not special in any way, as long as there are enough integers left reserved for use by the protocol prin-

cipals. For example, an integer range of 1 for the ID type is useless, as this means there is only one ID that every principal would be forced to use. The value 0 also cannot be used, since I reserve 0 to represent *null*, or the absence of a message or message component.

Given these types, I now define in UPPAAL's global declarations the relevant information for the scenario that is available globally to all principals. Firstly, I assign some ID's to the two communicating principals.

```
int ID_A = 6, ID_B = 7;
```

The public keys of Alice, Bob, and the attacker are also publicly available integers that I assign. Note that the type of these is not relevant, since they are not exchanged or parts of messages but rather used only in encryption. The only requirement is that they must be unique to each principal. The corresponding private keys for each of these are of course not public, but I fix them all to be one-tenth of their public counterpart. The encryption and decryption functions are written to account for this, as mentioned in § 4.2.3.

```
int PK_A = 20, PK_B = 30, PK_C = 40;
```

As UPPAAL does not allow models to directly exchange data (see § 4.2.1), a global variable *message* in which models can read and write messages to is necessary.

```
int message;
```

Since message components will also need to be concatenated and parsed, there are a couple of useful variables I define. If for the sake of an example,

a principal wanted to concatenate the ID of Bob (0111) onto the ID of Alice (0110) the principal would firstly need to *bit shift* the ID of Alice to the left by 4 first, resulting in 01100000. The principal then applies a bitwise OR operation with this and the ID of Bob, resulting in 01100111. In UPPAAL, this is accomplished with the expression $(ID_A << shift)|ID_B$. The variable *shift* is fixed to be the length of each message component (in this case 4).

```
int shift = 4;
```

If the principal received the message 01100111 and wished to recover the ID of A from this message, the principal would need to perform a bit shift to the right by 4. To recover the ID of B, the principal performs a bitwise AND with the word 00001111. Therefore I define a global variable for this bit "unmasking" number, which in decimal is 15.

```
int unmask = 15;
```

In Figure 3.3, messages which are read by a principal are added to its knowledge set. To capture this, I simply instantiate private local integer variables that can store all the possible knowledge that can be gained from the messages involved in the protocol. In this way, when messages are received by a principal, the relevant local variables are updated. As mentioned before, when the variables are initialised they will have a value of 0 which is a reserved value across the whole model to represent *null*.

```
int plain[6];
int key[6];
int max = 6;
int index = 0;


int encrypt(int param1, int param2) {
  index+=1;
  if (index < max) {
    plain[index] = param1;
    key[index] = param2;
    return index-cryptoheader;
  }
  return 0;
}

int decrypt(int param1, int param2) {
  param1+=cryptoheader;
  if (param1<max && key[param1]/10 == param2) {
    return plain[param1];
  }
  return 0;
}
```

Figure 4.4: Cryptography Code for NSPK with Movement in UPPAAL

4.3.2 Cryptography

Secondly, the principals must be able to use cryptography. Since the type range of ciphertexts is between 1 and 5, the arrays of plaintexts and keys must be defined with 6 slots to store all these potential ciphertexts. Figure 4.4 shows the UPPAAL code which implements encryption and decryption.

The *encrypt* function works as follows. The value of *index* is incremented, and if this new value is less than the length of the cryptographic arrays, then

encryption can be performed. As explained in § 4.2.3, the plaintext is stored at the index *index* in array *plain*, and the corresponding key that was used to encrypt this plaintext is stored at the same index in array *key*. The value of *index* becomes the ciphertext and is returned. The reason that *cryptoheader* is subtracted from index is due to my attacker model; in order to implement restriction (3) *Shallow Cryptography* in § 4.2.4. The attacker only attempts a decryption on a negative number.

```
int cryptoheader = 6;
```

Lastly, the principals must all be able to generate nonces (capturing the calculus rule *FreshMsg*), or numbers used only once in a run of a protocol. As mentioned in § 4.2.4, all nonces are typed. Figure 4.5 shows the code that allows the principals to generate nonces. Every principal uses the same function to generate nonces, and each generated nonce must be fresh and have never been used before. As mentioned previously, the nonce types are of the integer range $8 \leq \text{Nonces} \leq 15$. The *fresh* function embeds the *FreshMsg* rule from Figure 3.3.

4.3.3 Models of the Initiator, Responder, and Attacker

Now that the rules for movement and message passing have been embedded, the protocol specification for the initiator role of NSPK can be modelled in UPPAAL, as seen in Figure 4.6.

In Figure 4.6, the initiator of the NSPK protocol first generates a fresh nonce by calling the *fresh*() function from Figure 4.5. Next, they encrypt the

```
//Nonce Creation
int i = 7;
int maxNonce = 15;
int fresh() {
  if (i<maxNonce) {
    i+=1;
    return i;
  }
  return 0;
}
```

Figure 4.5: Nonce Creation for NSPK with Movement

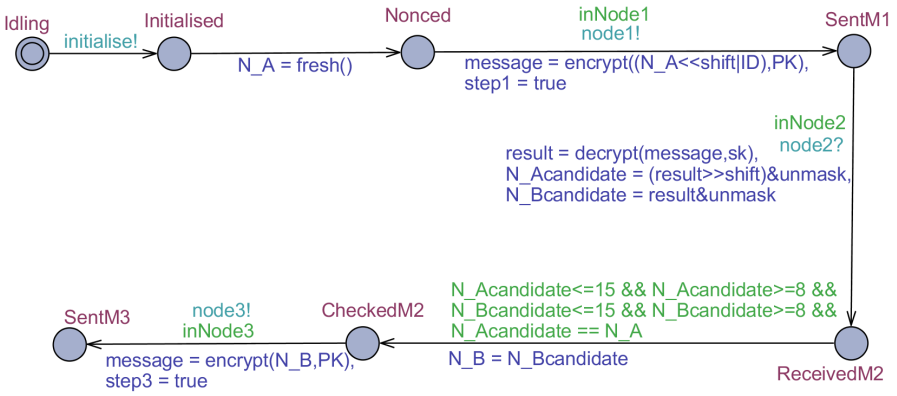

Figure 4.6: The Model of the NSPK Initiator

concatenation of this fresh nonce with their ID, with the public key of the principal they wish to talk to, which could be the honest responder principal or a dishonest attacker (this is modelled by a separate UPPAAL model shown in Figure 4.7). This message is written to the global *message* variable.

More specifically, *any* message can only be sent if the initiator is in the correct position (as mentioned in § 4.3.3), which is governed by the boolean parameters *boolParam1*. This means this transition is only possible if *boolParam1* is

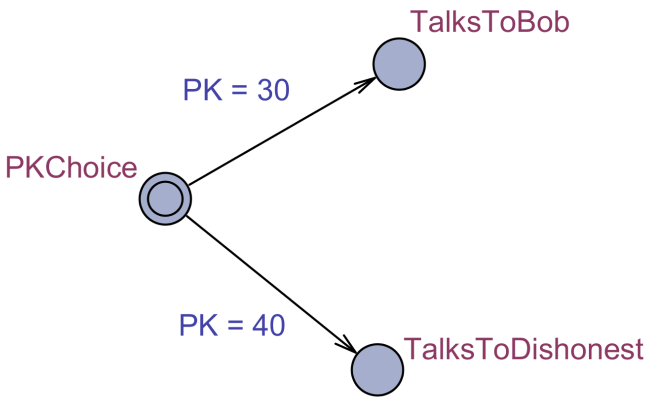

Figure 4.7: A Choice of Keys for the Initiator

set to True via movement when the initiator is instantiated (see § 4.3.3). The A global boolean value *step1* is set to true, signifying that the first message of the protocol has been sent. This will become useful for verifying security properties later (see § 4.3.4) and for use by the attacker to make sure they do not overwrite their own knowledge, which could cause an explosion of the state space and potentially missed attacks when the protocol is verified and analysed. Once the initiator has sent this message, they must wait for the reply, as per the protocol specification. The initiator may need to move to the correct position in order to receive a reply, so this transition is guarded by *boolParam2*. When this reply comes, the initiator decrypts it with their private key and the message components contained in the received message are assigned to "candidate" values ($N_ACandidate$ and $N_BCandidate$). They then type check the candidate values and check that the nonce $N_ACandidate$ is the same as the nonce N_A that they initially created. If $N_BCandidate$ type checks, then the initiator sends this nonce back encrypted with the same public key as in the first message. The initiator may need to move again to

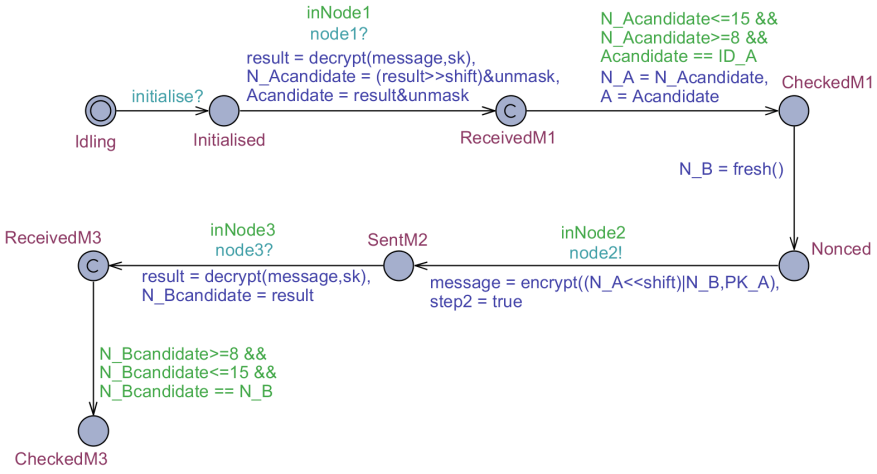

Figure 4.8: The UPPAAL Model of the NSPK Responder

send this message, so this transition is guarded by *boolParam3*.

The message passing of the responder is given in Figure 4.8. When they receive the first message, the responder decrypts it using their private key. The message components contained in the result of the decryption are stored in candidate values in order to be type checked. If the type check succeeds, the responder creates a nonce N_B and concatenates it onto the N_A received from the initiator, encrypting it with the initiator's public key. They then wait for the final message. When they receive it, they type check it and check if it is the same nonce that they created and sent in the previous message.

Lastly, the attacker brain is shown in Figure 4.9.

This attacker brain is an *implementation* of the general attacker schema in Figure 4.3. The general mode of operation is as follows. The attacker reads in a message on any of the available channels, which are only available if the attacker agent has moved to a specific location (governed by *boolParam1* and so on, similar to the initiator and responder). The received message is stored

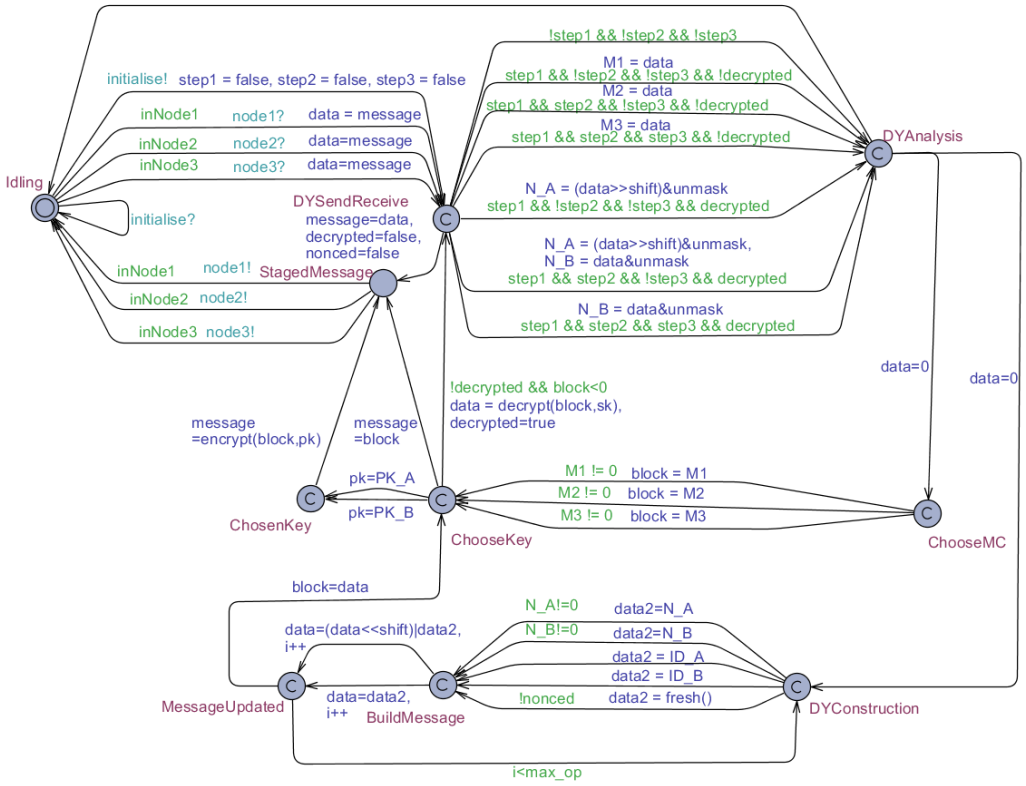

Figure 4.9: The UPPAAL Model of the Attacker

in a variable *data*. Depending on what step of the protocol the principals are in, the received message in *data* is stored in either *M1*, *M2* or *M3* (the "M" standing for message; there being three messages in total for NSPK). The attacker is now in the *DYAnalysis* state at this point, and can choose to construct a new message from its existing knowledge, replay a message, or decrypt and analyse a received/eavesdropped message *M1*, *M2* or *M3*. If the attacker chooses the latter, it loads the message into variable *block*. A "block" is one or more message components that are concatenated together which are either about to be decrypted and analysed or about to be sent (or encrypted and sent). The attacker is now in the *ChooseKey* state and can either choose

to "stage" the block for sending, pick a key to encrypt the loaded block and stage this encrypted block for sending, or decrypt the block with its private key (other keys that they might now could be added as necessary). In a situation where composed keys must be used, the attacker model will need to be adjusted to account for this, as my current attacker model does not handle composed keys. If the attacker chooses to decrypt, the attacker can update its knowledge set by recovering the message components (via bit-shifts and unmasking) from the decrypted message and storing them in locally defined variables.

Returning back to the *DYAnalysis* state, the attacker could choose to construct a message by transitioning to the *DYConstruction* state. The attacker can load a message component from its knowledge into variable *data2*, to be concatenated onto whatever is already stored in *data*. The variable i controls the maximum length of a message. The attacker can keep concatenating message components until $i == max_op$, after which, whatever is stored in *data* becomes a new block (as per restriction (2) *Message Length*) and the attacker returns to the *ChooseKey* state.

Instantiating the Scenario

In this section, one example NSPK scenario will be instantiated. Alice is a hub and Bob is made up of three nodes that are far apart in an environment (i.e. Alice might be a a fast-moving drone moving through each of the three nodes that make up "Bob"). In the environment is a single attacking hub. Figure 4.10 shows a diagram of this scenario.

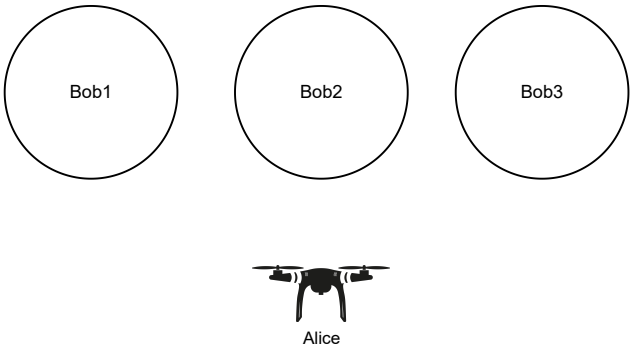

Figure 4.10: An Example Scenario of NSPK with Movement

$$
\begin{aligned}
\mathsf{Alice} := \; & A_{single}(Bob_1).\nu(N_A) \\
& \overline{Bob_1}\langle\{N_A, A\}_{K_B}\rangle.D_{single}(Bob_1).A_{single}(Bob_2). \\
& Bob_2(\{N_A, N_B\}_{K_A}).D_{single}(Bob_2).A_{single}(Bob_3). \\
& \overline{Bob_3}\langle\{N_B\}_{K_B}\rangle.\ 0
\end{aligned}
$$

Figure 4.11: A Trace of Alice in The Calculus

The three Bobs all share the same message knowledge set at any given time t, K^t_{Bob}. However, Bob_1 is responsible for receiving the first message in the protocol specification namely $\{N_A, A\}_{K_B}$. Bob_2 is responsible for replying with $\{N_A, N_B\}_{K_A}$, and Bob_3 is responsible for receiving the final message of the protocol, $\{N_B\}_{K_B}$. The environment for this scenario is given by $Env = (\{Bob_1, Bob_2, Bob_3\}, \{\})$ but again, other possible environments can be considered (more in § 4.4).

A possible execution trace of Alice in this scenario of this protocol using my calculus from § 3 could be as shown in Figure 4.11.

Alice moves to the relevant node to send / receive each message.

The corresponding trace for Bob is much simpler, since Bob is made up of three separate nodes each with their own default channels. These can be seen

$$\mathsf{Bob_1} := \quad Bob_1(\{N_A, A\}_{K_B}).0$$

Figure 4.12: A Trace of Bob_1 in The Calculus

$$\mathsf{Bob_2} := \quad \nu(N_B).\overline{Bob_2}\langle\{N_A, N_B\}_{K_A}\rangle.0$$

Figure 4.13: A Trace of Bob_2 in The Calculus

in Figure 4.12, 4.13, and 4.14.

Movement and The Environment

In order to capture movement among the principals, I firstly embed the environment into UPPAAL as a transition system from which all the hubs (including the attacker) can be instantiated in order to achieve individual movement in the environment. This UPPAAL model is depicted in Figure 4.15. Referring to § 4.2.2, the boolean parameters *boolParam1*, *boolParam2*, and *boolParam3* are values that monitor the location of the hub, and the relevant boolean values for the instantiated hub, are passed as parameters to the model upon instantiation.

Instantiating Alice

Alice in this scenario will be the initiator, and therefore for Alice to send and receive messages I instantiate an instance of the initiator model. Since Alice is a hub, as mentioned in § 4.3.3, I pass the boolean parameters *AliceInBob1*, *AliceInBob2*, and *AliceInBob3* to *boolParam1*, *boolParam2*, and *boolParam3*

$$\mathsf{Bob_3} := \quad Bob_3(\{N_B\}_{K_B}).0$$

Figure 4.14: A Trace of Bob_3 in The Calculus

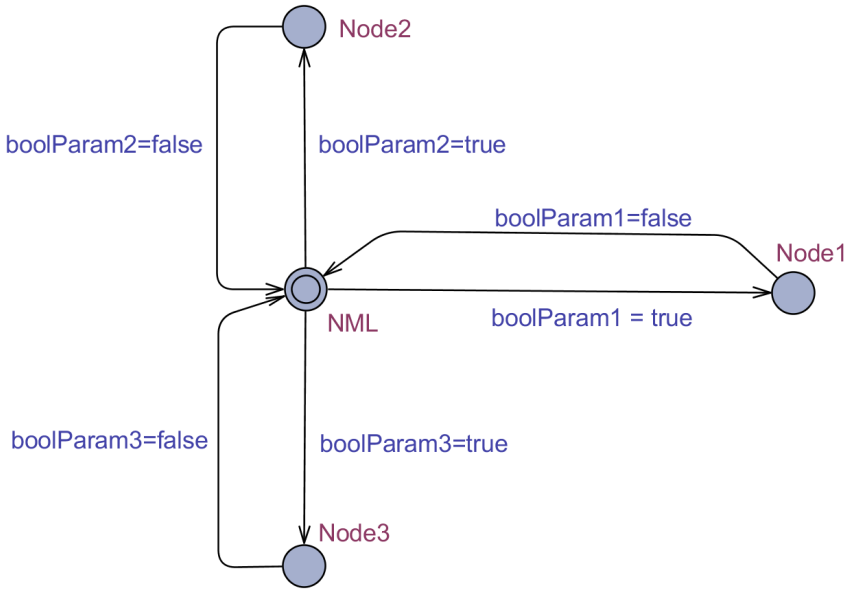

Figure 4.15: UPPAAL Model of the Environment

respectively, which all specifically describe Alice's location.

```
bool AliceInBob1, AliceInBob2, AliceInBob3;
```

Alice's possible movements can be captured quite easily in UPPAAL by simply instantiating the environment for her to freely move around in, again with boolean parameters *AliceInBob1*, *AliceInBob2*, and *AliceInBob3*. Any hub involved in any protocol is to be instantiated in UPPAAL this way. Their possible movements are captured by an instantiation of the environment that tracks their location with globally defined boolean values, and their message passing capabilities are captured in an instance of the initiator or responder

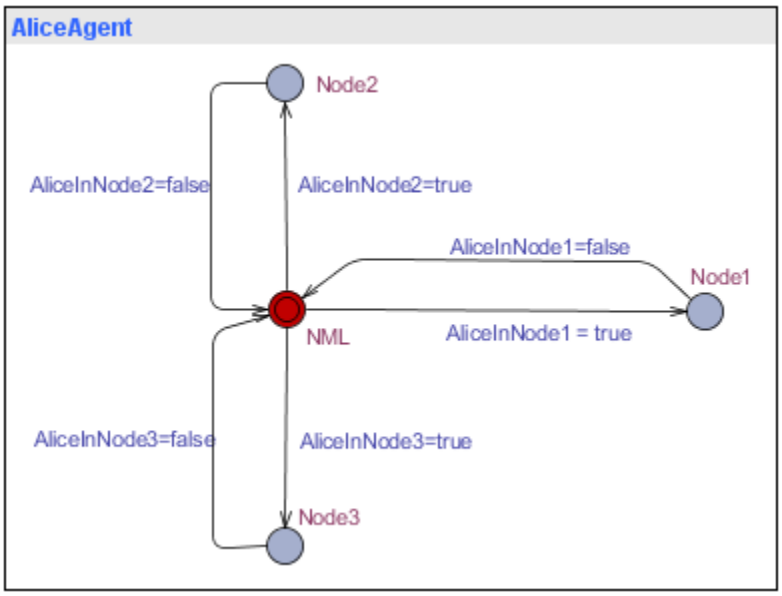

Figure 4.16: The Instantiation of the Movement Part of Alice

models.

Since Alice is made up of two parts, the movement and the message passing, Alice is instantiated in UPPAAL with the following statement in UPPAAL.

```
AliceAgent =
    Environment(AliceInNode1, AliceInNode2, AliceInNode3);
AliceBrain =
    Initiator(AliceInNode1, AliceInNode2, AliceInNode3);
```

When the model of the environment is instantiated for Alice, it produces the transition system depicted in Figure 4.16.

And the instantiated message passing part of Alice is given in Figure 4.17

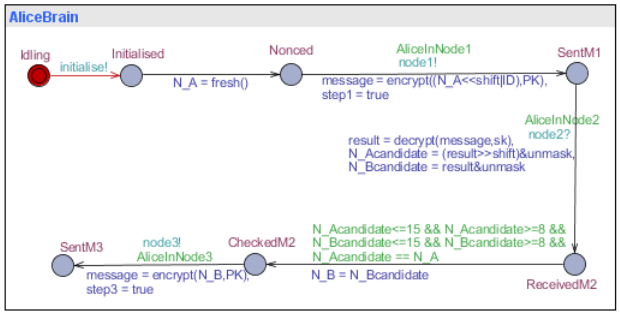

Figure 4.17: The Instantiation of Message Passing Part of Alice

Instantiating Bob

As mentioned in § 4.3.3, Bob is made up three nodes in the environment who act as the *responder*. Since Bob does not have to move to send a message, the guards *boolParam1* (and so on) can simply all be instantiated with the value *true* since nodes never need to move to send messages. Due to a how variables are instantiated in UPPAAL, I need to assign the value of true to a named global variable before passing i as a parameter when creating an instance of a node. This I have named *isNode*.

Therefore, Bob is instantiated via the following statement in UPPAAL.

```
BobBrain = Responder(isNode, isNode, isNode);
```

This produces the following instantiated model in Figure 4.18

Instantiating The Attacker

The attacker in this scenario is a dishonest hub who also moves around the environment (however, a dishonest node would be easy to instantiate for other analyses). So, I instantiate an environment for the attacker to move around

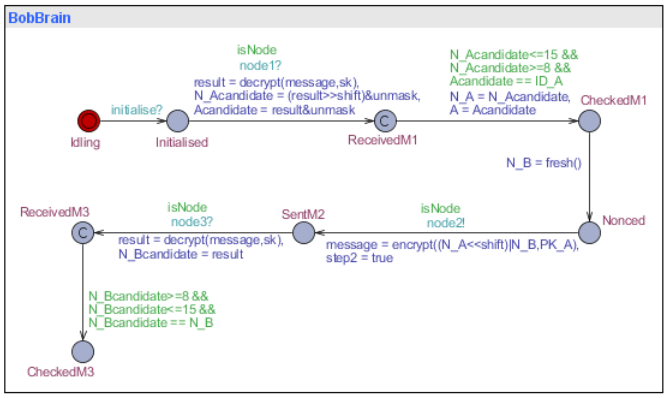

Figure 4.18: The Instantiation of Bob

in as well as an instantiation of the attacker brain.

```
AttackerAgent =
    Environment
    (
    AttackerInNode1, AttackerInNode2, AttackerInNode3
    );

aBrain =
    AttackerBrain
    (
    AttackerInNode1, AttackerInNode2, AttackerInNode3
    );
```

Which produces the following instantiated models in Figures 4.19 and 4.20.

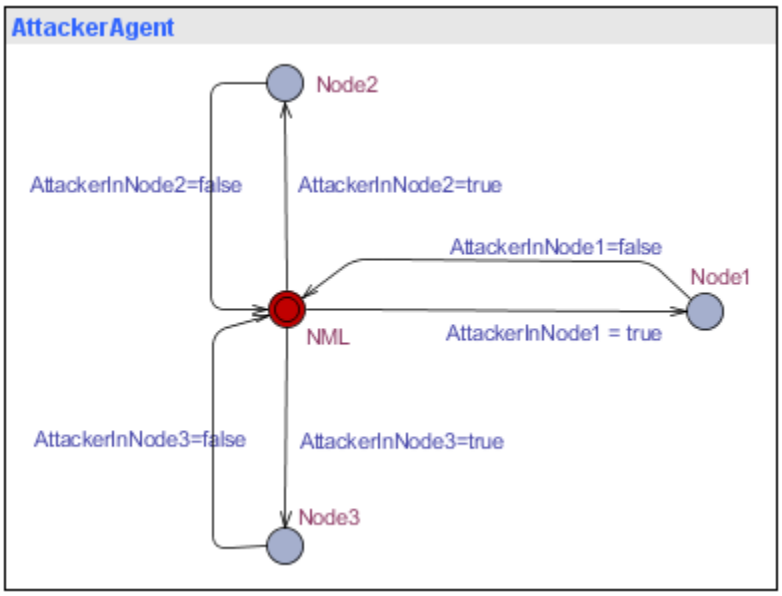

Figure 4.19: The Instantiation of the Movement Part of the Attacker

Table 4.1: Temporal Properties in UPPAAL

Name	**Property**	**Equivalent to**
Possibly	$E\langle\rangle\ p$	
Invariantly	$A[\ ]\ p$	$not\ E\langle\rangle\ not\ p$
Potentially Always	$E[\ ]\ p$	
Eventually	$A[\ ]\ p$	$not\ E\langle\rangle\ not\ p$
Leads to	$p \dashrightarrow q$	$A[\ ]\ (p\ imply\ A\langle\rangle\ q)$

4.3.4 Analysis and Verification

The UPPAAL tool allows one to check properties of a constructed system by means of *queries*, specified using UPPAAL's *requirement specification language*. As shown in Table 4.1, taken from the UPPAAL documentation, there are five temporal properties that can be employed within a query to reason about the system.

The "possibly" property $E\langle\rangle\ p$ evaluates to *true* for a timed transition

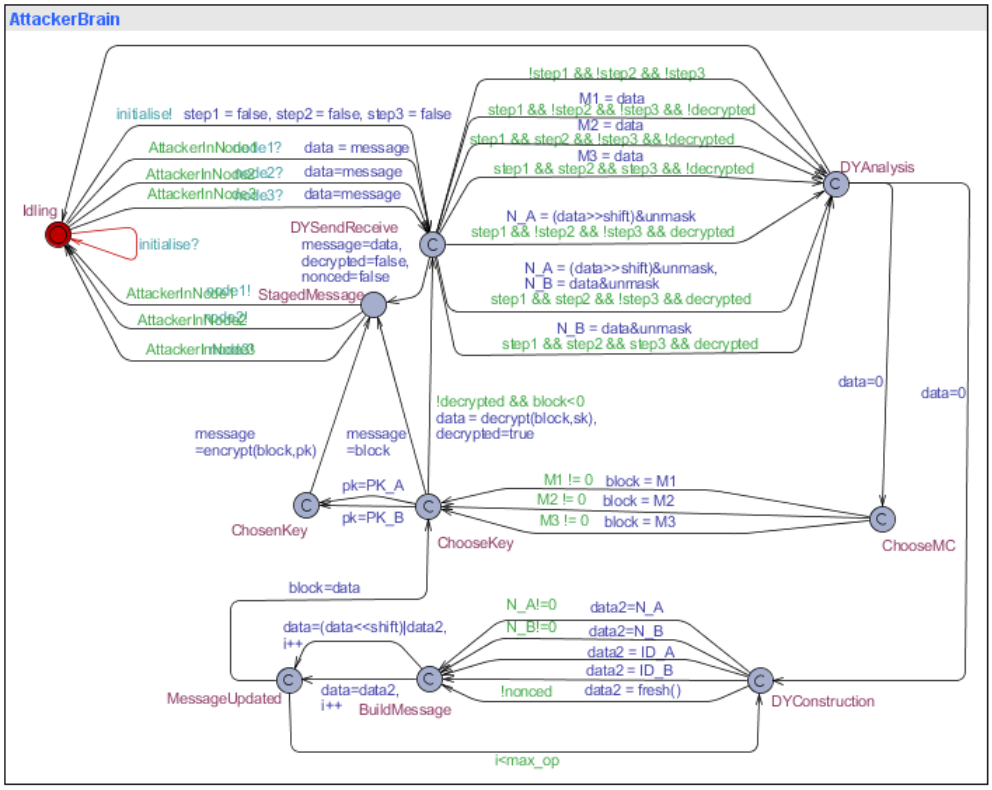

Figure 4.20: The Instantiation of the Message Passing Part of the Attacker

system if and only if there is a sequence of transitions $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n$ where s_n satisfies p. The temporal property I will use for all the queries is the "leads to" property, which denotes that whenever p holds, another state property q will eventually hold. All the security properties that I give below and in the case studies in § 5 primarily deal with authentication. Therefore the use of "leads to" here means that the UPPAAL verifier evaluates that if a formula expressing authentication criteria holds in a certain point in a trace, then another formula holds in a previous state of that trace.

Leads to is a combination of: (1) the "invariantly" property $A[]p$, which evaluates to *true* iff every reachable state satisfies p, (2) implication given by

syntax word *imply*, and (3) the "eventually" property $A\langle\rangle\ p$, which evaluates to *true* if (and only if) all possible transition sequences eventually reach a state satisfying p.

As mentioned in § 2.1.5, the NSPK protocol aims to establish mutual authentication. To this end, the I verify the protocol with respect to authentication properties for the purposes of this example. Each query in the rest of this section formalises an authentication property. Each one is input into the UPPAAL verifier, which will terminate either with a verification that that property holds, or a diagnostic trace of an attack which shows that this security property does not hold.

The following security properties are not necessarily to demonstrate the resilience to attacks of NSPK as a protocol, NSPK's vulnerabilities are already well documented. However, this exact same approach will be used in § 5.1 and § 5.2 where resilience to attacks particularly relating to movement and location are more interesting. For now, I simply show UPPAAL and my approach to verification and analysis in action.

Security Property 4.3.1 (Alice Aliveness)

$BobBrain.CheckedM3 \dashrightarrow\ !AliceBrain.Idling$

Security Property 4.3.2 (Bob Aliveness)

$AliceBrain.SentM3 \dashrightarrow\ !BobBrain.Idling$

Firstly, I verify some basic aliveness properties for the protocol. Security Property 4.3.1 verifies the aliveness of Alice, and states that if the Bob UPPAAL model is in the state labelled *CheckedM3*, then Alice must have initiated

the protocol with Bob. Security Property 4.3.2 similarly verifies the aliveness of Bob; if Alice has completed the protocol execution then it must have been Bob must have responded to Alice's messages.

In this scenario the tool finds Security Property 4.3.1 to hold. This is due to the fact that Bob always believes the initial message to come from Alice, so the attacker can never decrypt his reply, making completion of the protocol impossible if Alice is never involved. However, Security Property 4.3.2 trivially does not hold, as Alice can simply choose to carry out the protocol with the dishonest agent who themself has a public key. Indeed, the trace of this can be seen in Figure 4.21.

Security Property 4.3.3 (Nonce Agreement)

$$BobBrain.CheckedM3 \dashrightarrow$$
$$(AliceBrain.N_A == BobBrain.N_A and AliceBrain.N_B == BobBrain.N_B)$$

Security Property 4.3.3 states that if Bob is in the state *CheckedM3*, then the stored values for N_A and N_B for Alice must be the same as Bob's respectively. In this scenario, the tool terminates and finds this property to hold. However, take a scenario in which the attacker has obtained the private key of Alice. The attacker should be able to complete the protocol without giving Alice any knowledge of Bob's fresh nonce N_B. Indeed, UPPAAL finds that the attacker can. The trace of this attack can be seen in Figure 4.22.

Security Property 4.3.4 (Weak Authentication)

$BobBrain.CheckedM3 \dashrightarrow PK == 30$

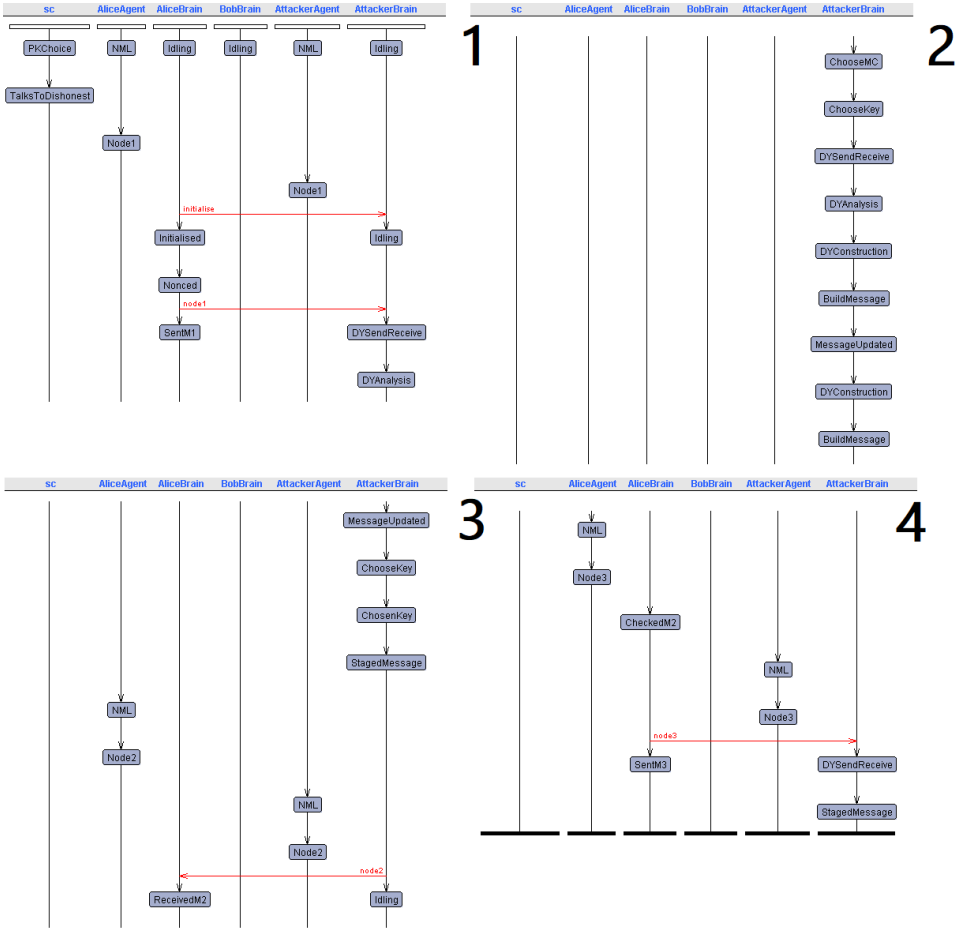

Figure 4.21: A Trace of a Possible Attack on Security Property 4.3.2

Security Property 4.3.4 states weak authentication, in that if Bob has finished running the protocol, then the key used to encrypt the messages by the initiator Alice in the communication must have been Bob's public key. The key PK is chosen by the instantiated model of *ScenarioChoice* which sets the public key that the initiator Alice will use for the duration of the protocol. Essentially, the property is verifying that if Bob has reached the end of the protocol, then *ScenarioChoice* must have set PK to 30 prior to this. However,

UPPAAL finds an attack on this property and provides a trace of the attack which resembles Lowe's attack except containing the movements of both the honest principals and the attacker themself. This trace is given in Figure 4.23.

The trace of the attack only shows the trace of UPPAAL system states that each principal are in after every transition, so the exact messages being sent cannot be seen in the diagram itself as they could in a standard message sequence chart. However, in UPPAAL one can inspect the values stored in the global and local variables (such as *message*) in order to see the details of the attack.

As seen in Figure 4.23, Alice talks to the dishonest principal so therefore uses the attacker's public key. Alice moves into the first node of Bob, and creates a nonce. The attacker also moves close to Alice by moving into the first Bob node and receives the message from Alice. Alice moves out of the first Bob node, and the attacker builds a message containing Alice's nonce and name. The attacker encrypts this message with Bob's public key and sends it to Bob. Bob, having type-checked this message creates a nonce and encrypts this nonce and Alice's nonce sent by the attacker with the public key of Alice. When the attacker moves into the second Bob node, the second Bob sends this message to the attacker. The attacker sends this message on to Alice who is also in the second Bob node. Alice moves to the third and final Bob node, type checks the last received message, and sends the final message containing Bob's nonce to the attacker (encrypted with the attacker's public key) who has also moved into the third Bob node. The attacker lastly sends the received nonce encrypted with Bob's public key to Bob.

This attack is reminiscent of Lowe's attack on NSPK, the man-in-the-middle attack. However, for this attack to work, the attacker has had to "chase" Alice as she moves through the three Bob nodes such that the attack is executable.

This result demonstrates the power of UPPAAL as a security property verification tool and of modelling security protocols with movement in this way. It shows that in an environment where principals can move anywhere, the tool finds violations of security properties with respect to the protocol specification in a memory-efficient and quick manner. Using unsophisticated timing techniques, I found that the tool takes approximately one second to verify each of these security properties on a Hewlett-Packard Pavilion x360 running Windows 10 Home 64-bit with an Intel Core i5-6200 2.30GHz processor and 8GB of RAM.

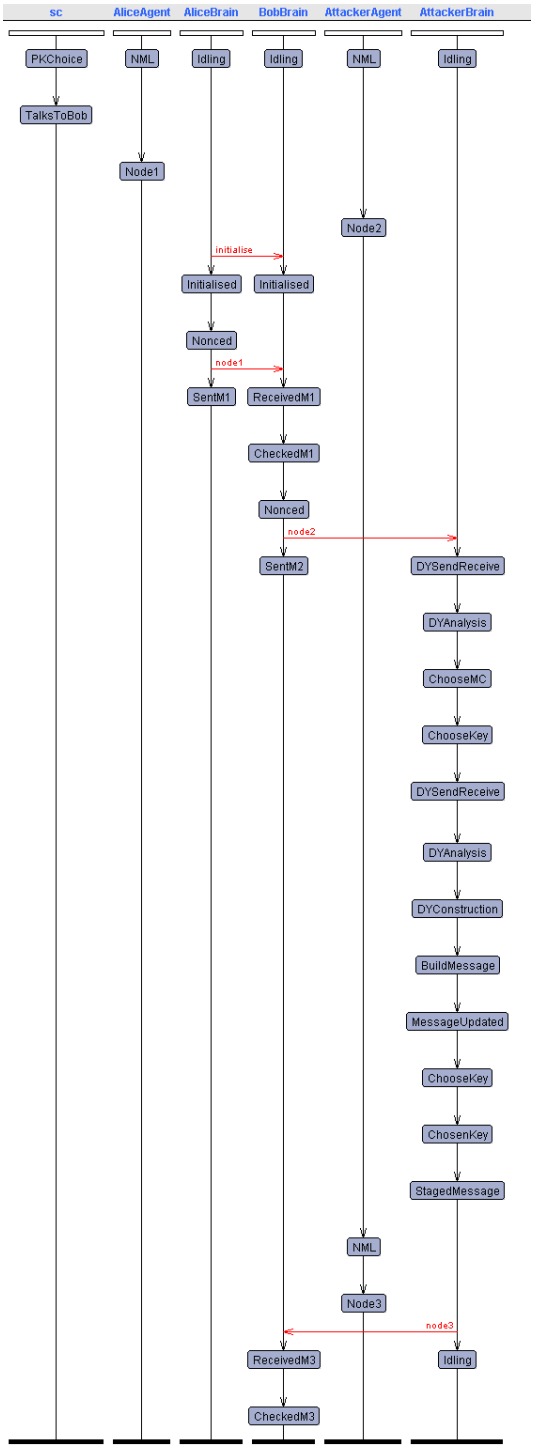

Figure 4.22: A Trace of a Possible Attack on Security Property 4.3.3

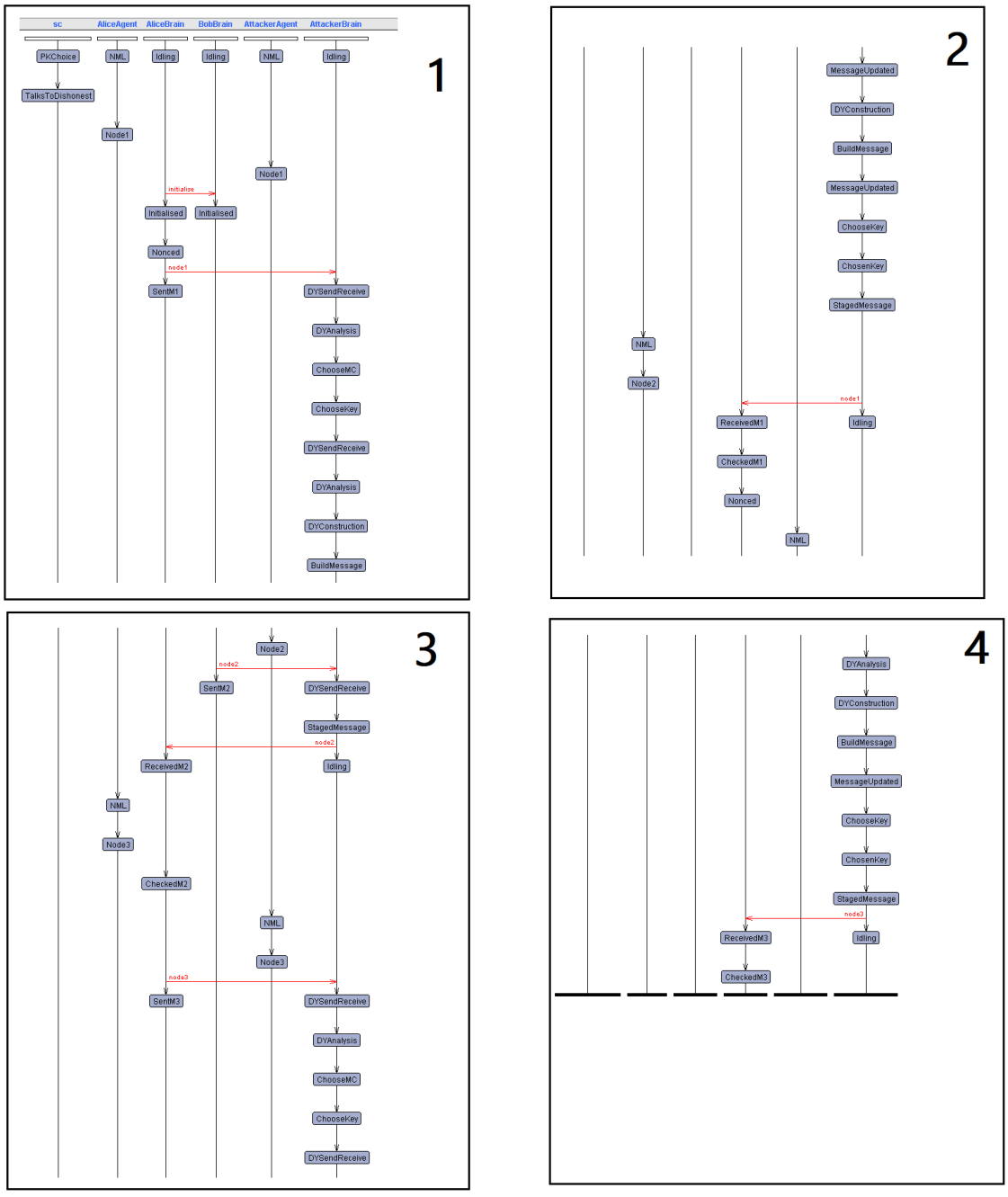

Figure 4.23: A trace of an attack on Security Property 4.3.4

4.4 EnvGen: Another Layer of Automation

When one creates an UPPAAL file that models a security protocol with movement, one will have defined various locations in which the protocol agents can move to or from. The environment is thus given by the arrangement of locations; whether they overlap or are far apart. In order to conduct an analysis for a security protocol in a different environment, an UPPAAL user would need to manually create each new UPPAAL file with alternative environments and analyse each one seperately. In order to save one from having to do this, I created *EnvGen*. EnvGen is a tool that takes a pre-existing UPPAAL file, generates all the possible environments from the locations defined in the original environment, and creates separate stand-alone UPPAAL files containing each possible environment. In EnvGen, the user can also input an UPPAAL query file and verify if the security properties one has defined within it hold across each generated UPPAAL file, all at once. The aim of developing this tool is for one to see how the environment directly impacts the attack traces, or more specifically, the attacker's movement.

4.4.1 Motivations

In § 4 thus far, I have only analysed the NSPK protocol in an artificially created environment, for the purposes of showing the efficacy of UPPAAL as a tool to approach the problem of analysing security protocols with movement. However for any such case study of a protocol being used in an environment, the environment can be instantiated in a number of ways.

I briefly touched on this concept in § 4.2.5. Due to the geometry of the model for an environment, there are a number of different ways an environment made up of multiple locations can be arranged in terms of the overlappings of the nodes with one another. In particular, the number of well-formed environments that can be constructed from n nodes is $2^{\frac{n(n-1)}{2}}$. It is important, for a full analysis of a protocol that involves moving between locations, that one should be able to analyse security properties for the protocol across all possible environments.

In reality however, $2^{\frac{n(n-1)}{2}}$ different unique environments will not need to be analysed. This is due to the fact that the environments defined in § 3.3 in reality are *planar graphs*, where the graph's nodes represent environment nodes and the edges represent an intersection of two nodes. These graphs must be planar, or else graphs where edges cross over each other would lead to geometrically impossible environments. Kuratowski's theorem [27] states that there can be no complete planar graphs for $|\phi| > 4$. This implies that there are no environments with more than five nodes in which all nodes overlap with each all the other nodes. Another way to imagine this is that I simply exclude generation of any environment in which $|O| > 6$, since complete planar graphs of 4 nodes require 6 edges as can be seen in Figure 4.24. Although none of my case studies contain more than five nodes, this would reduce the space of environments to analyse drastically for complex environments.

EnvGen is written in Java and features a graphical-user-interface. Although EnvGen is still in development and testing as a stand-alone extension tool for analysing security protocols with location and movement in UPPAAL,

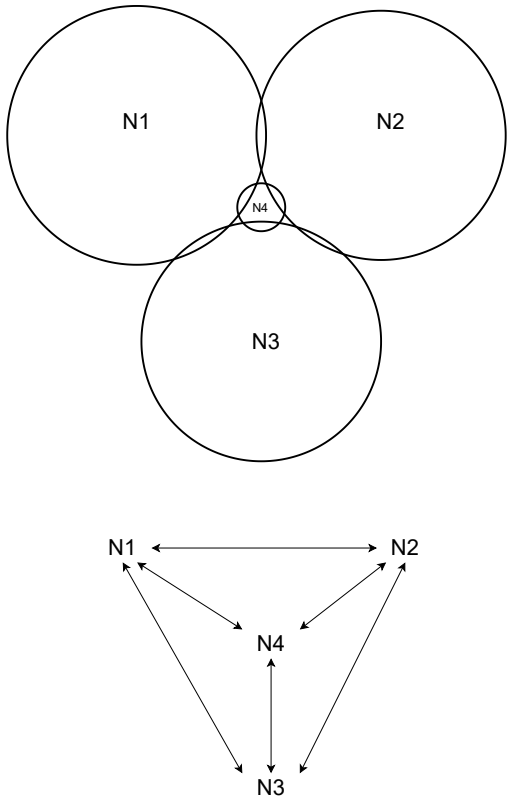

Figure 4.24: A Comparison of an Environment and its Equivalent Planar Graph

I have been able to use it for analysis on all possible environments not only for the NSPK example in § 4.3, but also in the New-Bank-Card case study in § 5.1. EnvGen is not required for the case study in § 5.2 since the environment here contains only one node. § 4.4.2 shows how to use EnvGen to obtain analysis results for all possible environments, using the NSPK with movement example from § 4.3.

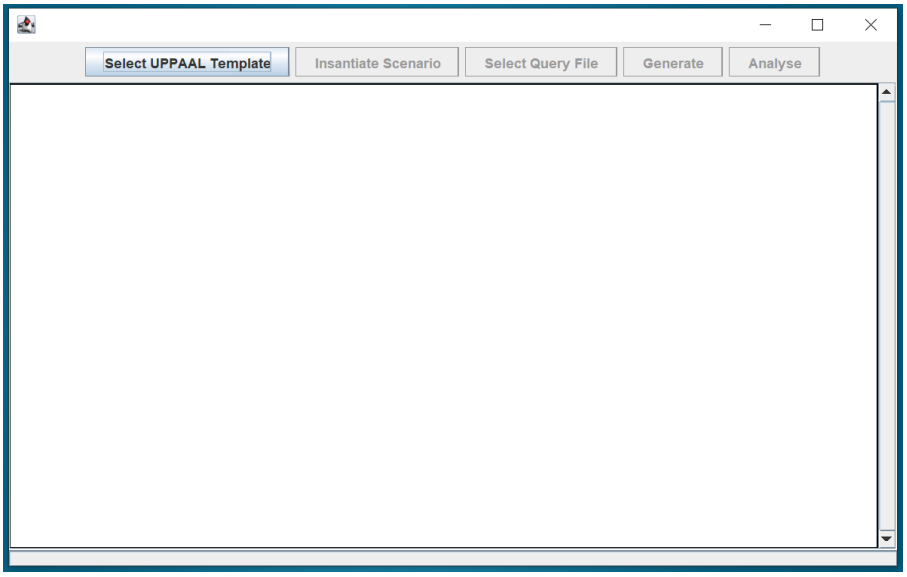

Figure 4.25: The User Interface of EnvGen

4.4.2 Using EnvGen and Obtaining Results

As an example, this section shows the use of EnvGen to generate all possible environments from the scenario given in § 4.3 and how a full analysis and diagnostic traces for all found attacks are stored, organised, and examined in UPPAAL.

When the user starts EnvGen the main user interface appears. This is seen in Figure 4.25. The main controls for the user are selecting an UPPAAL template to carry out analysis on, instantiating a scenario (i.e. which principals are nodes and which are hubs), selecting a query file containing security properties for analysis, generation of all possible environments, and carrying out analysis on all possible environments. Below these controls is the main console to which relevant information is output to the user.

When one selects an UPPAAL template, they can browse their local ma-

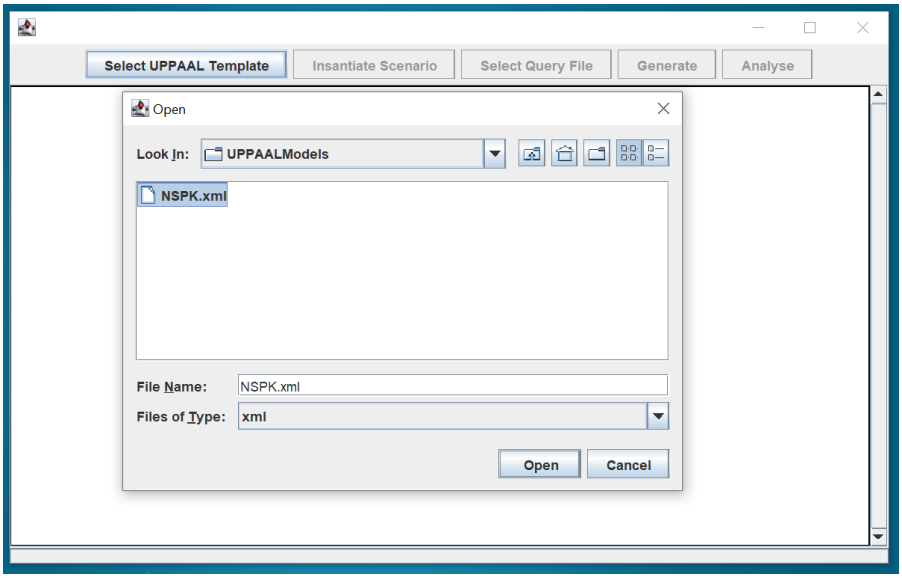

Figure 4.26: Selecting a Pre-existing Model in EnvGen

chine for an UPPAAL model, which are stored as XML files. In this example, I choose the NSPK with movement UPPAAL file. This can be seen in Figure 4.26.

When an UPPAAL file is selected, EnvGen displays the name of the selected file. EnvGen also reads the list of locations in the pre-existing model of the environment and displays these in the console. Since the NSPK with movement example features three nodes who all represent Bob (see § 4.3.3), these three nodes, *Node1*, *Node2*, and *Node3* are displayed in the console.

The user is now ready to instantiate a scenario. This allows one to decide which principals are mobile hubs and which are stationary nodes, including the attacker. All the user has to do is enter the names that they would like the initiator, responder, and attacker to have respectively. A user instantiating the same scenario of NSPK with movement as in § 4.3.3 is shown in Figure 4.27,

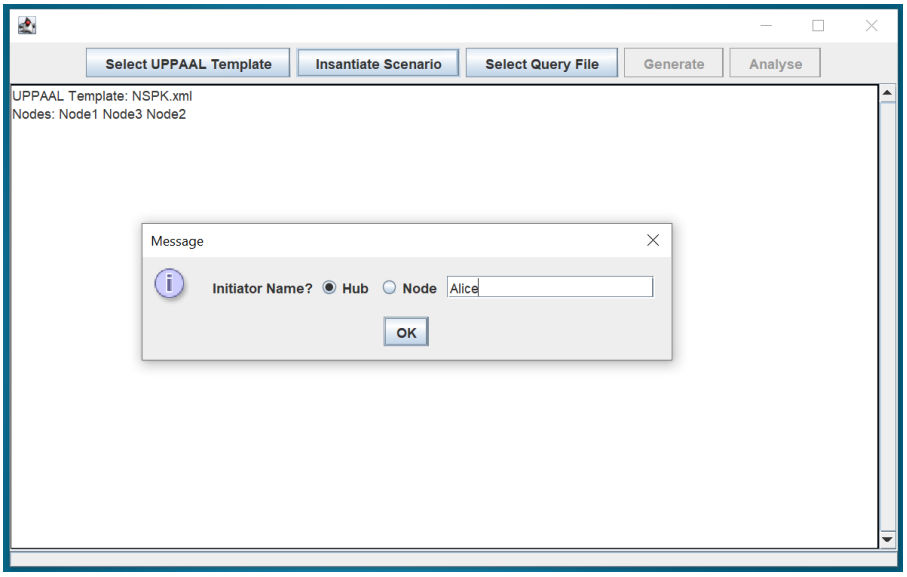

Figure 4.27: Instantiating Alice

4.28, 4.29.

Once the scenario has been instantiated, the user must now load a query file containing all the security properties given in 4.3.4. Figure 4.30 shows another file explorer prompt allowing the user to search for and load a query file.

The user can now generate all the possible environments from the locations found in the UPPAAL model. Given that in this model there are three locations, one expects EnvGen to generate 8 unique environments (by the formula mentioned in § 4.4.1. Note that this will include the environment that exists already in the original NSPK with movement example, where all three Bob nodes are far apart. EnvGen will create eight separate UPPAAL files whose models of initiator, responder, and attacker are identical to the selected pre-existing file, but the environment will be different. Each of these generated

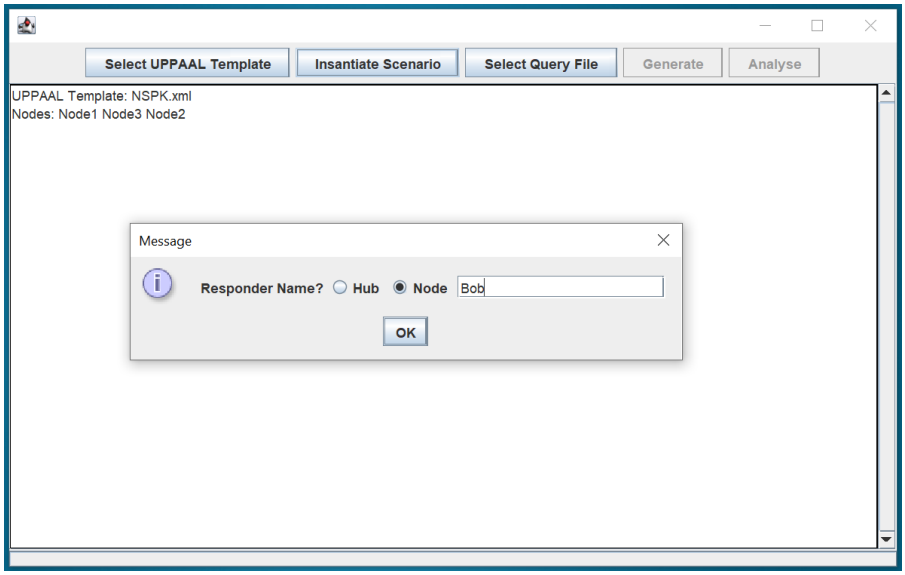

Figure 4.28: Instantiating Bob

UPPAAL files are ready for analysis immediately upon generation. EnvGen directly manipulates the XML in the UPPAAL files in order to create new unique environments. Once the environments have been generated they are stored together in a folder.

The way in which EnvGen computes the possible environments is the following. First, it computes the number of overlaps that could exist, which is given in this case of 3 nodes by $\frac{3(3-1)}{2} = 3$. EnvGen then simply takes a binary string of this length 3, 000, and begins counting up to 111. Each 1 bit in the string represents an overlapping that occurs between two of the nodes in the environment (or rather, an edge in the planar graph of the environment). Thus, simply by iterating through all combinations of 1's and 0's in the binary string by incrementing the binary string, one obtains all combinations of overlaps. Besides this, it is simply a matter of manipulating the XML to produce

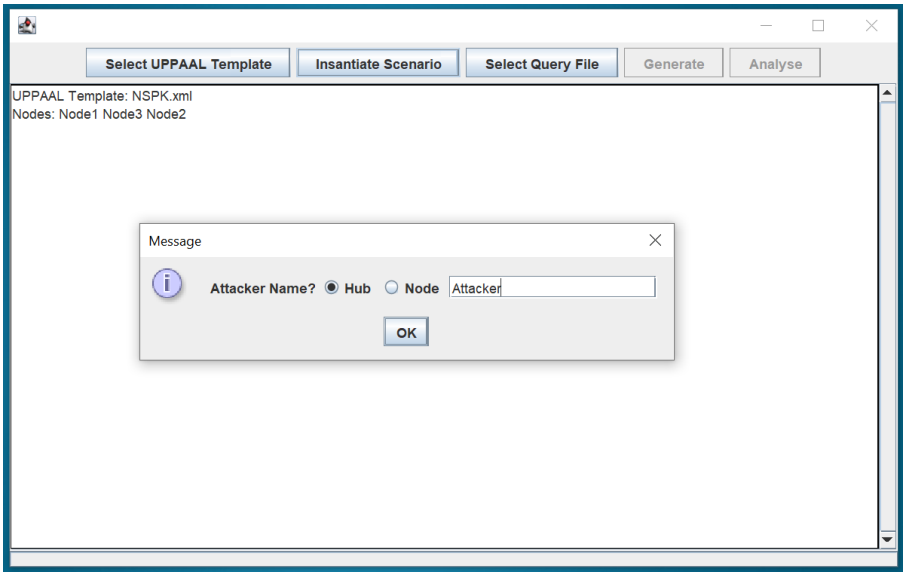

Figure 4.29: Instantiating Attacker

the environments themselves. All the files that are generated from the NSPK wth movement example are shown in Figure 4.31.

Figure 4.32 shows one of the environments generated by EnvGen. In this environment, all of the nodes overlap with each other. This means that a hub can be in $Node1$ and $Node2$'s overlap, $Node1$ and $Node3$'s overlap, or $Node2$ and $Node3$'s overlap. I assume that a mobile hub's trace of movement is "normalised" in the sense that if $Node1$ and $Node2$ overlap, the mobile hub will always move into the overlapping region instead of moving between the separate nodes, which would be redundant movements. As a result of this, only the locations of the overlaps need to be modelled.

Now that these files have been created, the user can analyse the security properties given in the selected query file across all of the generated files at once by simply clicking *Analyse* in the user interface. EnvGen creates a batch file at

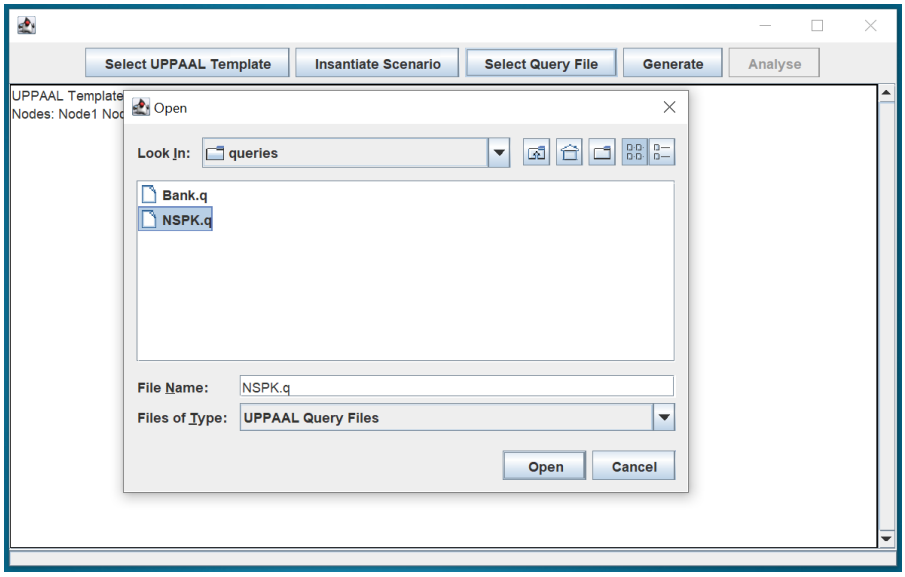

Figure 4.30: Selecting a Pre-Existing Query File

runtime and writes commands to carry out the analysis of each UPPAAL file one by one. The batch file contains instructions which call the command line analysis tool *verifyta* and passes each UPPAAL file, along with the selected query file, and options to produce diagnostic traces (should an attack be found) all one by one, as parameters to the tool. Figure 4.33 shows the EnvGen tool carrying out the analysis across all the generated UPPAAL files. The EnvGen console is simply echoing everything that would appear on the command line should the user be using *verifyta* in a typical shell.

Now that the analysis is complete, the user can inspect the diagnostic traces that *verifyta* has output using the UPPAAL tool itself. If the user were to load *environment7.xml* which is the environment depicted in Figure 4.32, the user can load one of the diagnostic traces for this file into UPPAAL and inspect the steps of the attack trace. Figure 4.34 shows an attack trace on Security

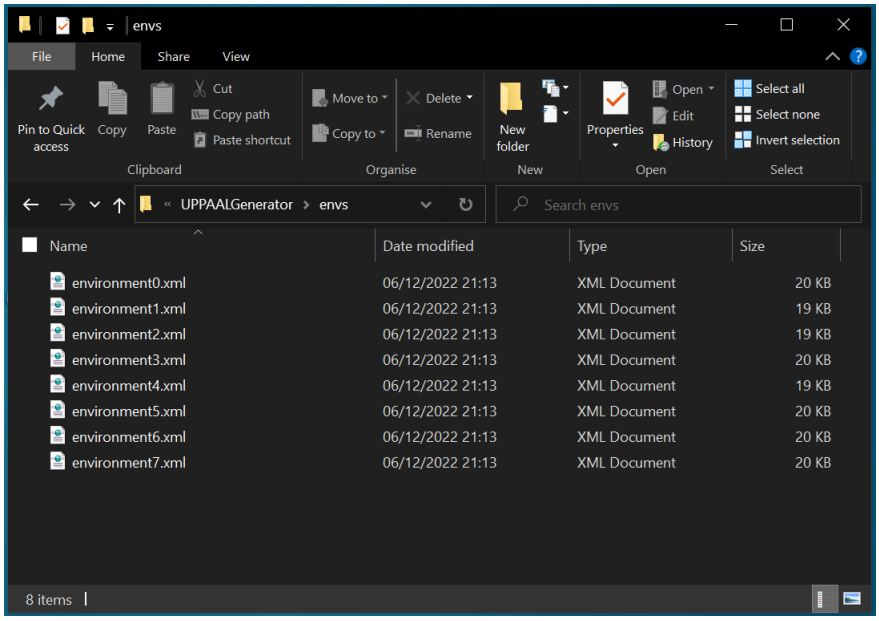

Figure 4.31: The Generated NSPK With Movement Files With Unique Environments

Property 4.3.4 in *environment7.xml* where the locations of the Bob nodes all overlap with each other. Naturally, in this attack trace there are fewer movements by *AliceAgent* and *AttackerAgent* needed, leading to a shorter attack trace. In this attack, the messages that the attacker creates and sends accomplish the same result as in Figure 4.23, however the movements differ. The attacker agent and the movement part of Alice move into the overlap between Bob_1 and Bob_3. The first message of NSPK is received by the attacker, and the attacker replays Alice's nonce and ID to Bob in the same fashion as the standard Lowe's attack from [30]. The attacker agent and the movement part of Alice then move to the overlap of Bob_3 and Bob_2. Here, the rest of the attack is carried out with no further movements required. Bob_2 replies accordingly to the attacker's message, and the attacker forwards this message

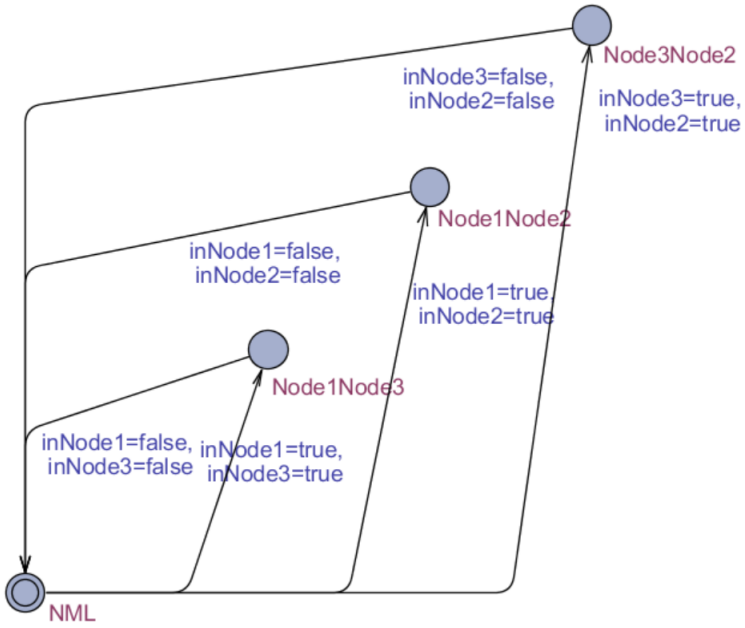

Figure 4.32: An EnvGen-Created Environment in Which All Nodes Overlap

on to the message passing part of Alice, *AliceBrain*. Alice responds accordingly to the attacker and the attacker finally forwards the final message of NSPK on to Bob_3 without the need to move.

As mentioned, EnvGen is still in development but allows me to analyse all possible environments for a given scenario. Not only this, but EnvGen provides some flexibility when it comes to instantiating principals as mobile or static. It remains to be seen what more can be done with EnvGen, such as instantiating multiple collaborating initiators, responders, or attackers, and flexibility with regards to the knowledge of the individual principals. In addition to this, the tool will need thorough testing and refinement of the user interface.

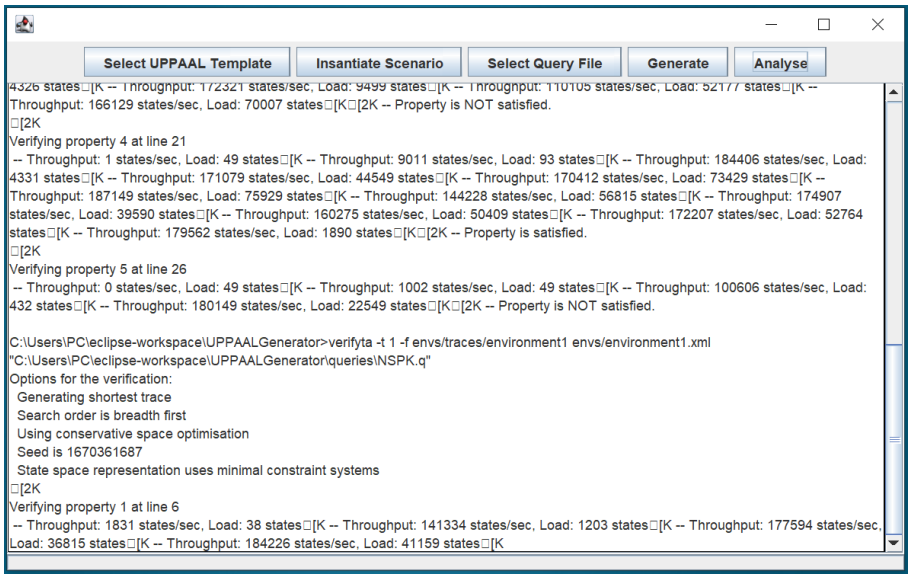

Figure 4.33: EnvGen Analysis In Action

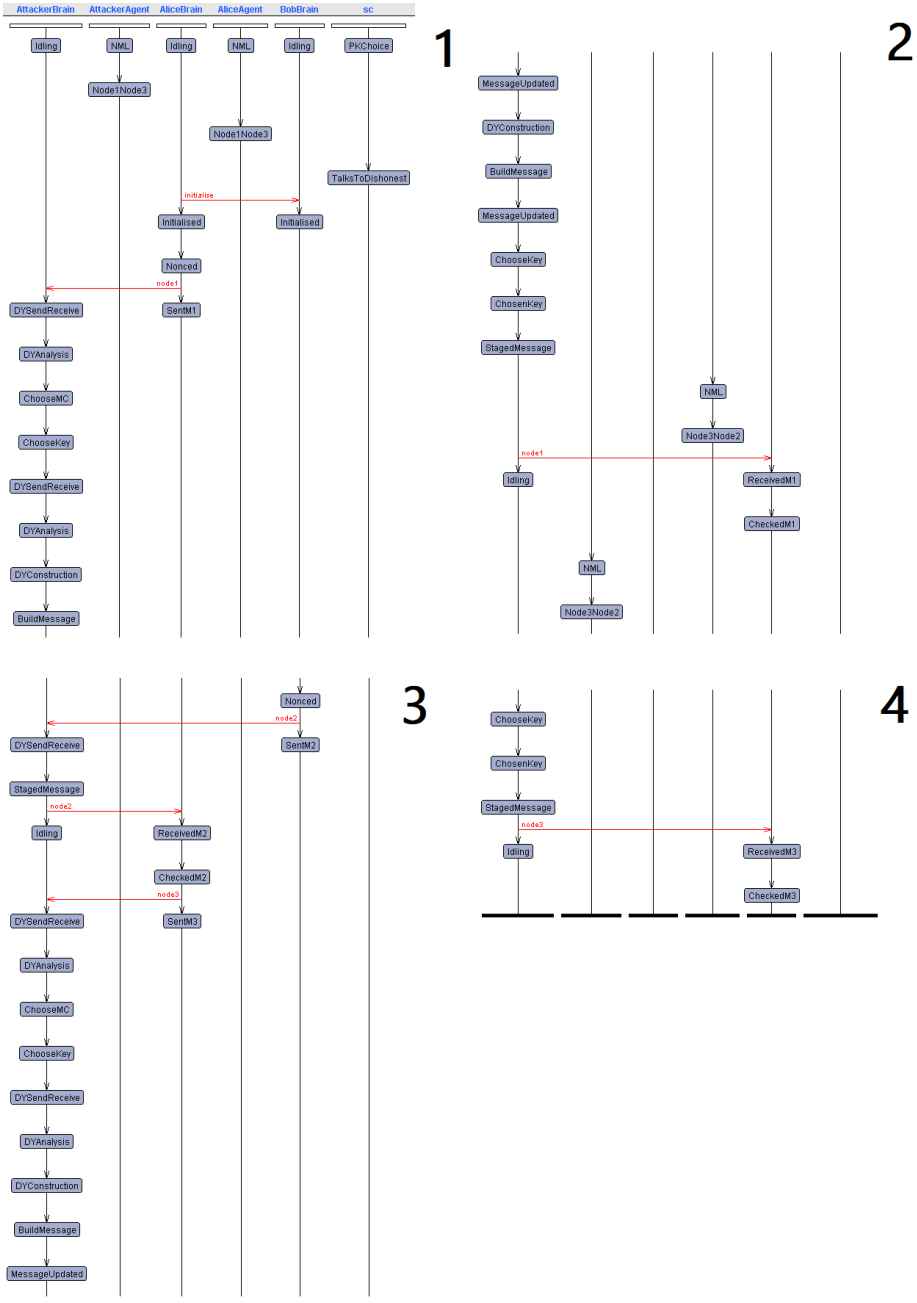

Figure 4.34: Example Attack on Security Property 4.3.4 in an EnvGen-Created Environment

4.5 Conclusions

In this chapter I have taken the rules from my calculus in § 3 and embedded them into the tool in the form of labelled transition systems and publicly available functions. Message manipulation, message passing, cryptography, movement, and fresh generation are all accounted for. I have also shown in this chapter how my calculus can model a simple example scenario of NSPK with movement. I have refined the attacker models from [26] and [18] in § 4.2.4 resulting in a new and improved attacker schema which can be implemented into UPPAAL for each case study. The general arrangement in UPPAAL of any security protocol with movement which is modelled consists of an initiator, a responder, an environment, an attacker, and a method by which certain choices are made by the initiator with regards to keys and so forth. From these models, a number of scenarios can be instantiated. I can choose who is a hub and who is a node by giving them a movement part by instantiating the environment for them, and the initial knowledge of each principal can be modified to produce a different scenario for analysis.

The example using NSPK with movement shows the approach in action, and I have explained how each calculus rule is embedded step-by-step through modelling the protocol in UPPAAL. I have shown how this protocol with the added dimension of movement can be analysed and how traces of attacks which depict movement of the principals and the attacker can be automatically found by verifying security properties that I give to UPPAAL.

As mentioned in § 1.2, there is nothing in my approach that would prevent it from being done in a tool such as Tamarin. However, UPPAAL fares

better when it comes to capturing movement, since transition system states easily model locations and transitions easily model movement between them. This is made even more apparent by the work carried out for EnvGen in § 4.4 through UPPAAL's API. Generation of new environments can be done automatically quite naturally by simply writing a program that manipulates UPPAAL's somewhat self-explanatory XML files for models.

EnvGen is currently still in the development phase, and future work would certainly be to finish development and testing. Currently for instance, one still needs to inspect traces by loading them into UPPAAL's simulator, but it would be better to be able to see traces without needing UPPAAL at all. I would need to build an interpreter for UPPAAL's trace files and be able to render them for view. Additionally, the console output in EnvGen is simply whatever the command-line verification tool *verifyta* echoes out. It would be preferable to have more readable feedback for the user to understand better what the program is doing.

Chapter 5

Case Studies

5.1 Case Study: The New-Bank-Card Protocol

In this section, the New-Bank-Card protocol from § 2.2.2 is modelled in UPPAAL to demonstrate further the power of the approach and to provide analysis and verification of a protocol which involves more complex movements and more involved principals.

For this case study, the only environment that will be considered is where every node is far away from one another. This is because if every node is far away, then it is likely the attacker will have to make at least one movement. In general *all* possible environments are considered (in § 4.4), but the aim is to show how my approach enables the analysis of security protocols with movement, rather than to carry out a thorough analysis of a specific protocol, regardless of whether it is a realistic protocol like the New-Bank-Card protocol and NSPK with movement, or a real protocol like the SegCom protocol in § 5.2.

This case study shows the versatility of the approach, in particular showing how surprisingly complex protocol scenarios involving many movements can be instantiated from just initiator, responder, environment, and attacker models that capture a protocol specification.

5.1.1 The Protocol in The Calculus

In this section, I first model the case study in adherence to the rules of the environment and calculus.

There are three nodes in the environment, the bank branch, the billing

address, and the ATM involved in the protocol. The current address of the customer is taken to be in no man's land as no communication takes place at this location.

Thus, $\phi = \{Branch, BillingAddress, ATM\}$ who all have associated default channels with the same names, the set of node processes is $\{\mathsf{Branch}, \mathsf{BillAdd}, \mathsf{ATM}\}$, and the set of hub processes is $\mathcal{H} = \{\mathsf{Cus}, \mathsf{Emp}\}$. Hereafter, "Bank" is the collective name for the set of nodes, and as such the ID of the Bank is given by B and the ID of the Customer is given by C. *CardDetails*, *NewCardDetails*, and *PIN* capture the old card number, the new card number, and the PIN of the customer, respectively.

The goal of the customer is to activate a new card. This involves requesting, receiving, and activating the card. The ordering and delivering of a new card is handled by a *bank employee*, whose goal in the protocol is to authenticate the correct customer in person, having correct details on record, and giving the new card to the customer at the billing address. The goal of the ATM is to activate the new card once the customer has provided it with the correct details.

Figure 5.1 shows the message sequence chart of the protocol, which requires movement of the customer and the bank employee for messages to be sent and received. If this protocol is executed successfully, then all agents will have fulfilled their protocol goals.

What allows this case study to demonstrate the versatility of the approach in UPPAAL is that there are three nodes, only one of which performs any communication (the ATM), and the two hubs (the customer and the employee)

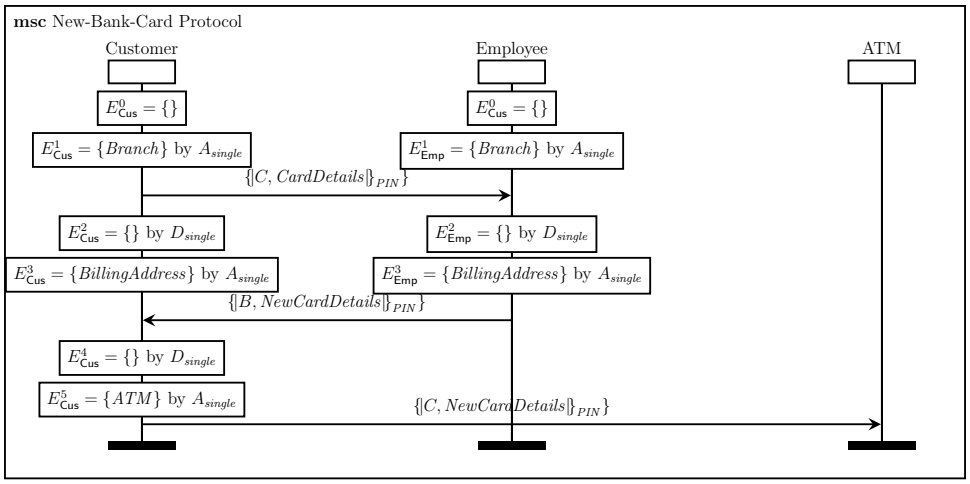

Figure 5.1: The New-Bank-Card Protocol

must "meet" at two of the nodes to exchange messages by means of the node's communication channel.

I formalise the protocol specification for the environment using the calculus given in § 3.

The trace of the customer in the calculus is

$$
\begin{aligned}
\mathsf{Cus} := \quad & A_{single}(Branch). \\
& \overline{Branch}\langle \{\!|C, CardDetails|\!\}_{PIN} \rangle. \\
& D_{single}(Branch). \\
& A_{single}(BillingAddress). \\
& BillingAddress(\{\!|B, NewCardDetails|\!\}_{PIN}). \\
& D_{single}(BillingAddress). \\
& A_{single}(ATM). \\
& \overline{ATM}\langle \{\!|C, NewCardDetails|\!\}_{PIN} \rangle.\ 0
\end{aligned}
$$

Here one can see the customer moving in order to send and receive messages

on the default channels of each node. The trace of the employee is

$$\begin{aligned}\mathsf{Emp} := \quad & A_{single}(Branch).\\ & Branch(\{\!|B, NewCardDetails|\!\}_{PIN}).\\ & D_{single}(Branch).\\ & A_{single}(BillingAddress).\\ & \overline{BillingAddress}\langle\{\!|B, NewCardDetails|\!\}_{PIN}\rangle.\ 0\end{aligned}$$

The trace for the ATM is simpler:

$$\mathsf{ATM} := \quad ATM(\{\!|B, NewCardDetails|\!\}_{PIN}).\ 0$$

The branch and billing address processes are simply a 0 (halt), since they do not send or receive any messages.

5.1.2 UPPAAL Implementation

The following subsections show how the techniques outlined in § 4.2 are instantiated for this case study.

Message Passing

By reference to the MSC of the protocol in Figure 5.1, there are only three types to consider: the ciphertexts, the IDs, and the card details. In the New-Bank-Card protocol, each message has no more than 2 message components per message. To be economical in my usage of the 15 bits that can be stored in the global variable *message*, I fix all message components to be 7 bits in length,

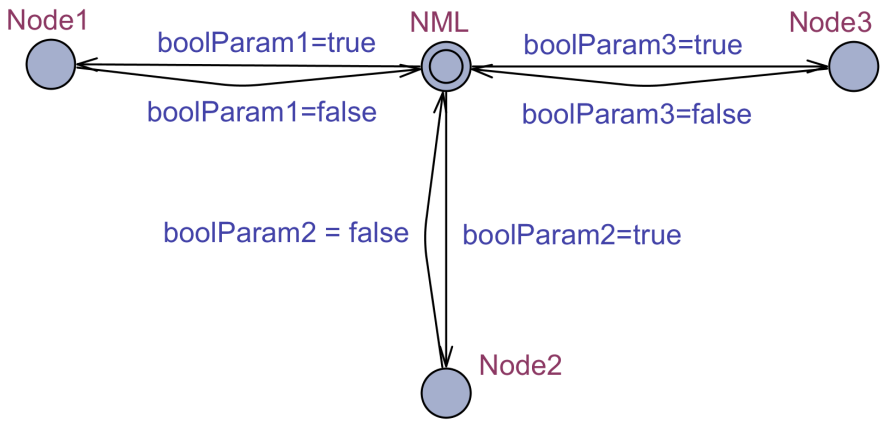

Figure 5.2: The Environment for the Scenario of The New-Bank-Card Protocol

and assign the following integer ranges to each type of message component: $1 \leq$ Ciphertext ≤ 7, $8 \leq$ ID ≤ 63, and $64 \leq$ Card ≤ 127.

The attacker brain can generate nonces using available functions, similar to § 4.3. For the scenario considered in § 5.1.4 the nonces are of type *Card*. The reason for this choice is that the bank and the customer are forming an agreement on a new variable of type *Card* by carrying out the protocol. The new card is effectively the only variable message component in the protocol from the point of view of the customer; IDs and PINs are constants.

Movement

In this protocol, the customer and the employee are hubs so I use UPPAAL to keep track of their location by way of two instances of the environment for each hub respectively, as was done in § 4.3.

The model for the environment is given in Figure 5.2.

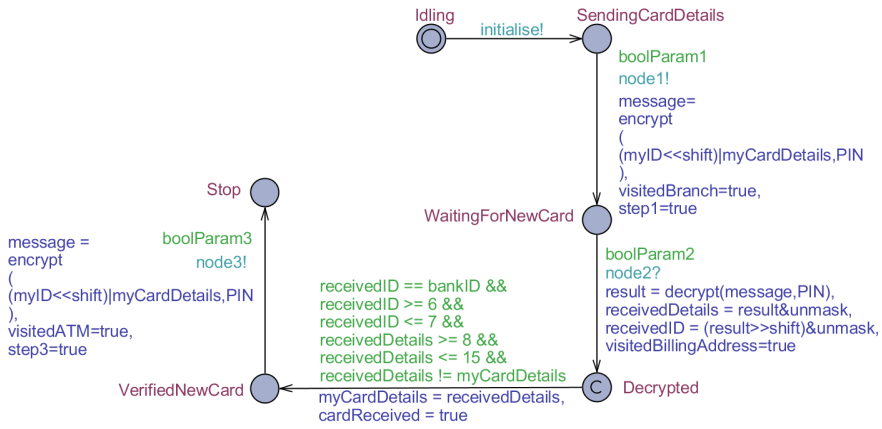

Figure 5.3: UPPAAL Model of the Initiator in the New-Bank-Card Protocol

Models of the Initiator, Responder, and Attacker

Following the convention and techniques from § 4.3, the customer will be instantiated as the initiator of the protocol, and the bank as the responder. Figure 5.3 shows the UPPAAL model for the message passing side of the initiator of the protocol.

In the model, the customer first builds an encrypted message consisting of their ID and card details, encrypted with their PIN. The ciphertext is written to *message*. The customer may need to move to the bank branch to send this message, therefore this first transition is guarded by *boolParam1*. The customer is now in the *WaitingForNewCard* state, and waits to receive the new card from the bank. The customer will need to move to the billing address to receive this message so this transition is guarded by *boolParam2*. Upon receiving this new card, the customer type checks each message component, and updates their card details accordingly. The customer moves to the ATM and sends the final message; their ID concatenated with their new card details

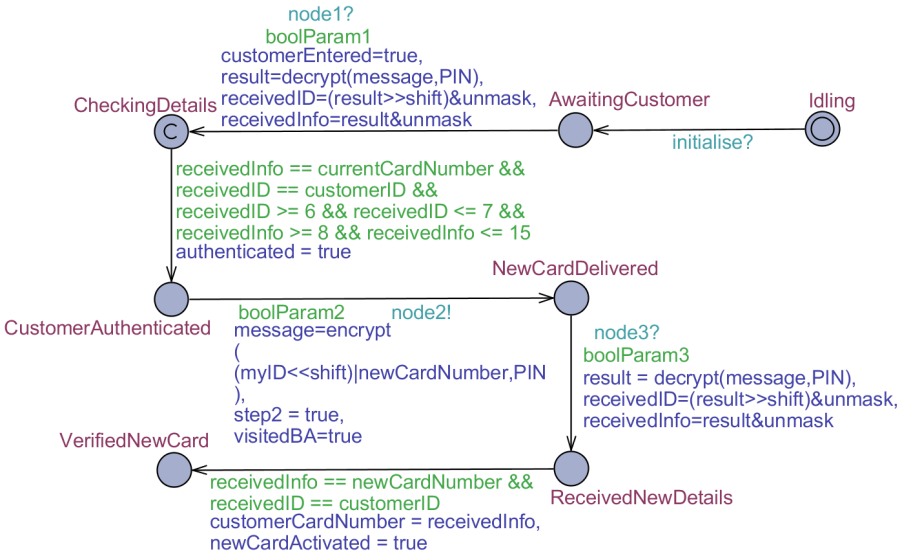

Figure 5.4: UPPAAL Model of the Responder in the New-Bank-Card Protocol

and encrypted with their PIN.

In Figure 5.4 the bank waits for the customer and employee to both arrive. When the customer has encrypted and sent their details, the employee receives them, decrypts and type-checks them. If they match their own records about that customer, the customer becomes authenticated. At this point, the bank encrypts its own ID with the new card details for the customer with the customer's PIN, and the employee delivers it to the customer after moving to the billing address, bringing the bank to the *NewCardDelivered* state.

The employee plays a special role in the protocol, which is to hand deliver the card to the customer. They first go to the bank branch node where they authenticate the customer. The employee, equipped with the new card details for the customer, leaves the branch and moves to the billing address where they drop off the new card to the customer who is also at the billing address. The customer simply waits at the billing address for the new card from the

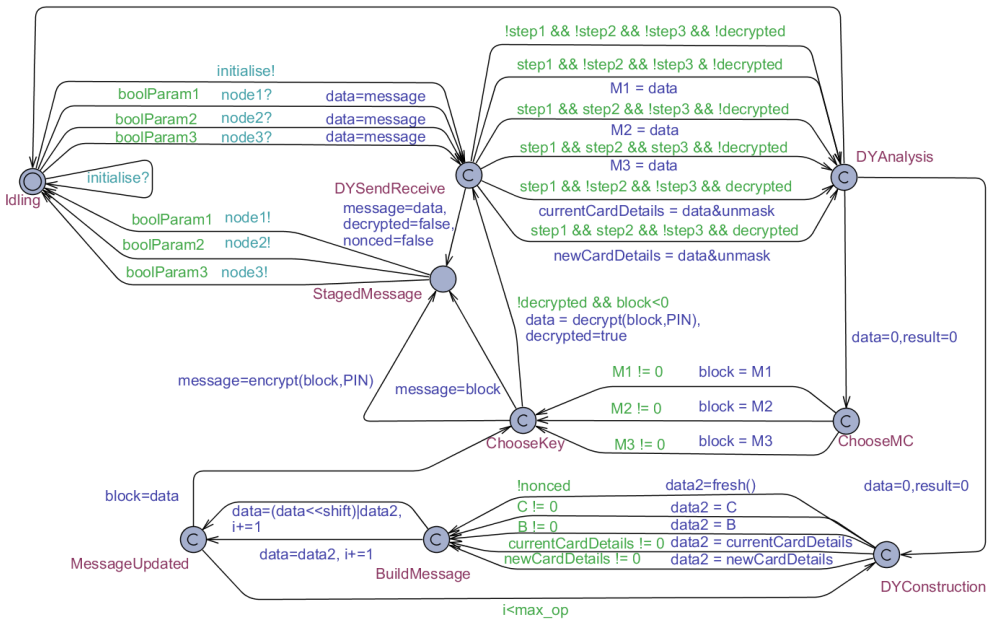

Figure 5.5: UPPAAL Model of the Attacker Brain in the New-Bank-Card Protocol

employee to be delivered when the employee moves there, or the employee waits for the customer to arrive at the billing address if they haven't arrived yet. The ATM waits for the new card details to be encrypted and sent to it by the customer once the customer has left the billing address. The ATM decrypts the new details and type-checks them. They compare them to the records the Bank has on file and either activates the new card or does not.

Finally, I instantiate the attacker agent and brain for this case study. The attacker agent can move anywhere in the environment much like in § 4.3. The attacker agent is an instantiation of the environment, and the UPPAAL model for the attacker brain can be seen in Figure 5.5. As the attacker agent moves, different communication channels become available to the attacker brain.

5.1.3 Instantiating the Scenario

For the analysis, I instantiate a single realistic scenario to demonstrate the approach. Consider one possible scenario within the environment, in which a dishonest person has been able to obtain the PIN number of another customer, possibly through a side channel attack of some kind, such as phishing. Other scenarios may be considered in future work, e.g., where the ATM and branch are the same location, the attacker brain does not initially know the PIN, or the customer and the bank all sign their messages with a digital signature using public key encryption. For now, I demonstrate the approach on a single scenario to show the efficacy of my approach in UPPAAL.

The customer is the initiator of the protocol and is a hub. The "bank", however, is made up of a node and a hub; the ATM is an automated system with no human present so I treat this as a node with which the customer must move to to communicate with the bank. Meanwhile, the bank branch and billing address are both indeed nodes, but the bank must send an *employee* (i.e. a bank teller / courier) to each location respectively for the messages to be received. For simplicity, I will consider the employee to be a single hub (i.e. the movement part of the "bank"). The environment itself is made up of the three nodes (being the bank branch itself, the billing address, and the ATM). The attacker is a dishonest hub. The customer and the bank share a symmetric key which is the customer's PIN, the value 34, which is also known to the attacker in this scenario as previously mentioned. These principals are instantiated as such in UPPAAL with the commands listed in Figure 5.6.

Instantiating the customer produces the transition systems pictured in Fig-

```
CustomerBrain =
  Initiator(CusInBranch, CusInBA, CusInATM, 34);
CustomerAgent =
  Environment(CusInBranch, CusInBA, CusInATM);

BankBrain =
  Responder(EmpInBank, EmpInBA, isNode, 34);
BankEmployee =
  Environment(EmpInBank, EmpInBA, EmpInATM);

AttackerBrain =
  Attacker
  (
  AttackerInBranch, AttackerInBA, AttackerInATM
  );

AttackerAgent =
  Environment
  (
  AttackerInBranch, AttackerInBA, AttackerInATM
  );
```

Figure 5.6: Instantiation of the Scenario for Analysis in UPPAAL

ures 5.7 and 5.8.

The Bank is instantiated to produce the transition system pictured in Figure 5.9, and the employee responsible for being present at the bank branch and delivering the new details to the billing address in depicted in Figure 5.10.

Finally, the attacker is instantiated in Figures 5.11 and 5.12.

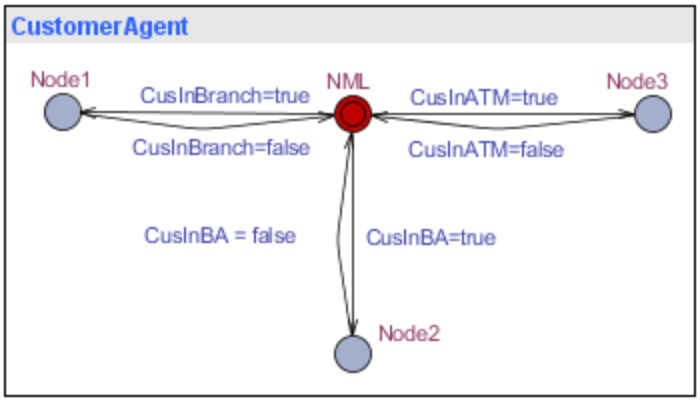

Figure 5.7: The Instantiation of the Movement Part of the Customer

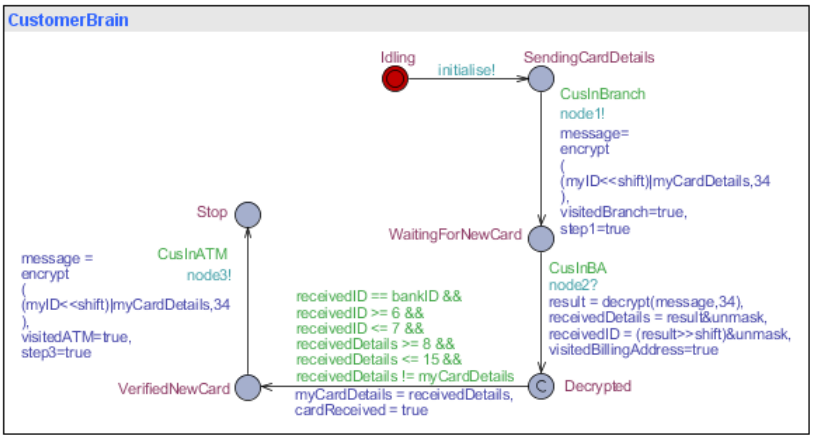

Figure 5.8: The Instantiation of the Message Passing Part of the Customer

5.1.4 Analysis

The initial state is as follows. The customer and employee start in no man's land, the bank is in its initial state (open and ready for customers to walk in), and the attacker knows in particular the following: the ID of the customer and the bank and the PIN of the customer. As I remarked, for this first case study I am only showing the versatility of the approach; I am not looking for attacks on this protocol, which actually quite trivially hold given the scenario

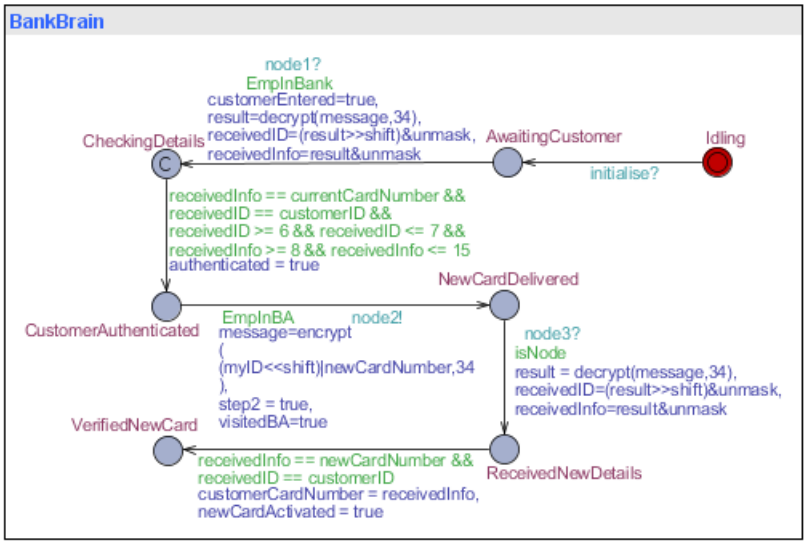

Figure 5.9: The Instantiation of the Bank

in which the attacker knows the customer's PIN number.

Location Verification

My approach allows me to reason specifically about where agents are at a certain point during the protocol execution. Boolean flags that are set to true during the execution of a protocol allow me to reason about the knowledge and location of the agents during the protocol execution. Important Boolean flags for the customer are *VerifiedNewCard* which is flagged as true after the customer receives and type-checks their new details, and *visitedBranch* which is flagged as true when the customer moves and goes into the branch. For the bank, important Boolean flags are *customerEntered* which is true when the customer moves into the bank branch, and *CustomerAuthenticated* which is flagged as true once the bank decrypts, verifies, and type-checks the customer's initial information. I can also reason directly about the location or state of

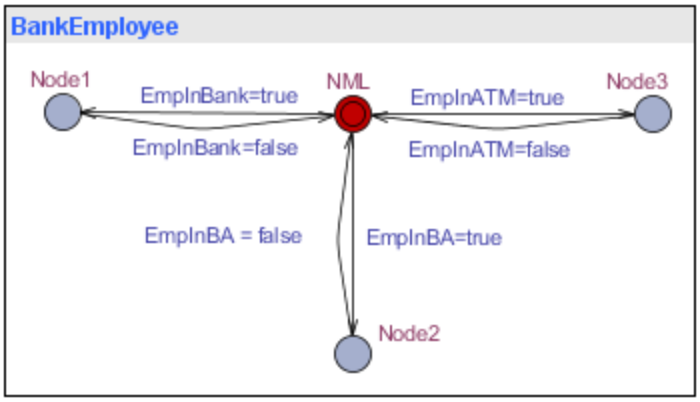

Figure 5.10: The Instantiation of the Bank Employee

the customer or employee. For instance, *CustomerAgent.Node1* means that the customer is currently in the branch.

As a first example, one might expect the new card to be delivered by a genuine bank employee, which is captured by Security Property 5.1.1

Security Property 5.1.1 (Employee Delivery)

$CustomerBrain.VerifiedNewCard \dashrightarrow BankBrain.visitedBA == true$

Security Property 5.1.1 states that if the customer has just received the card, then the employee must be at the billing address. This is of course falsified by the UPPAAL verifier, the simple explanation being that the employee does not have to authenticate themselves to either the bank or the customer so the protocol specification would need to be altered to prevent this attack. The attack is seen in Figure 5.13.

In this attack, the attacker simply moves to the bank branch (as does the customer) as one can see the *CustomerAgent* and *AttackerAgent* enter the *Node1* state. The customer initialises the protocol with the bank, but

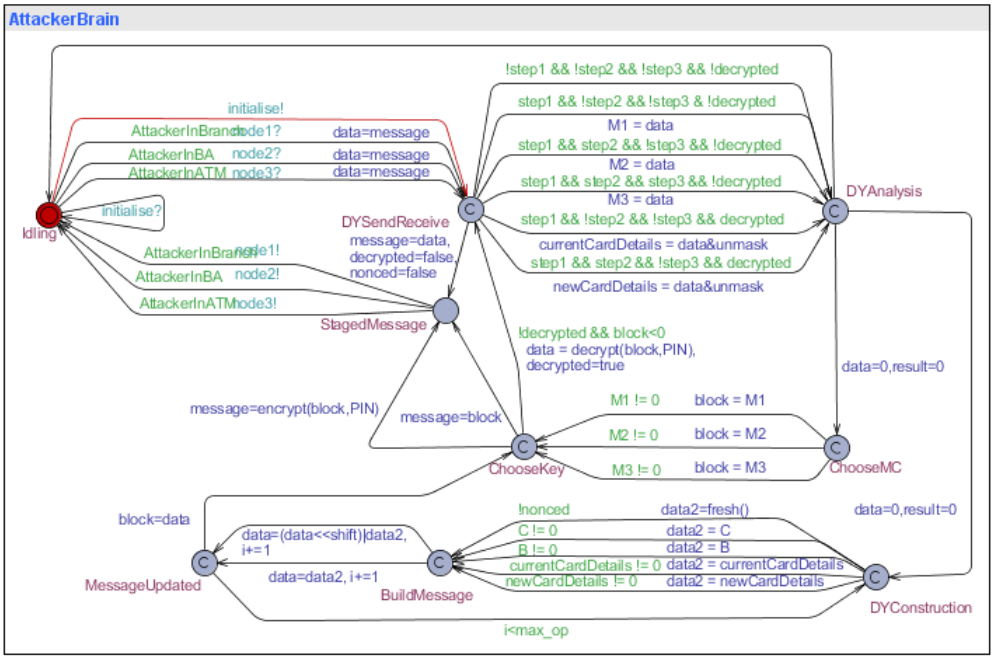

Figure 5.11: The Instantiation of the Attacker Brain

the customer gives the old card details to the *AttackerBrain* instead of the bank. When the customer meets with the *AttackerAgent* at the billing address (*Node2*), the *AttackerBrain* gives to the Customer a dummy card that the *AttackerBrain* created. This is only possible in this scenario because the attacker brain knows the PIN of the customer, which is required to build the message containing the new (fake) card details.

I next reason more rigorously about the protocol and how its goals reflect the movement and the location of the agents: Security Property 5.1.2 specifies that if the branch has authenticated the customer, then the customer must have visited the branch at some point.

Security Property 5.1.2 (Customer Authentication)

$BankBrain.CustomerAuthenticated \dashrightarrow CustomerBrain.visitedBranch == true$

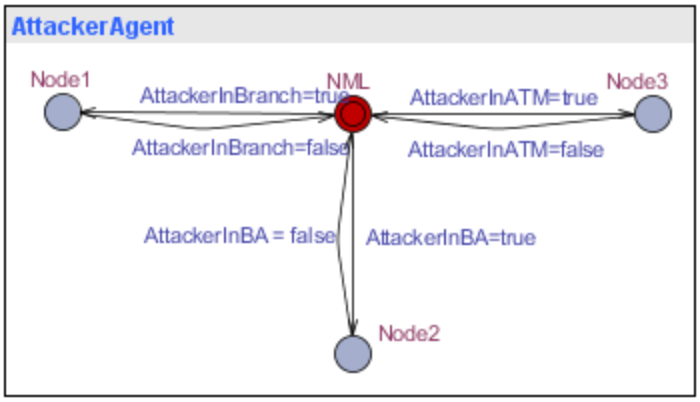

Figure 5.12: The Instantiation of the Movement Part of the Attacker

For this scenario, UPPAAL verifies that this property holds. However for other scenarios, for instance if the attacker initially knew the customer's card details, they would trivially be able to authenticate themself in place of the customer.

Aliveness

Next I analyse some aliveness properties in the protocol for the scenario under consideration, in particular between the customer and the bank.

Security Property 5.1.3 (Aliveness of Bank)

$CustomerBrain.VerifiedNewCard \dashrightarrow BankBrain.customerEntered$

Security Property 5.1.4 (Aliveness of Customer)

$BankBrain.authenticated \dashrightarrow CustomerBrain.visitedBranch$

In short, Security Property 5.1.3 says that if the customer has received a card, then the bank branch must have had the customer enter their doors.

When I analyse this property in UPPAAL, it finds a counterexample equivalent to the attack found in Figure 5.13.

Security Property 5.1.4 is verified to hold for this scenario. But take the scenario in which the attacker has been able to obtain the customer's old card details via a side-channel attack. If the attacker knows the customer's card details they could trivially authenticate themselves to the bank as the customer thus violating this property. A trace of this is shown in Figure 5.14. In this attack, the attacker simply moves into the bank branch (Node1), and initiates themselves to the customer by encrypting the customer's details and encrypting it with the PIN, sending the encrypted message to *BankBrain*. *BankBrain* reaches the *CustomerAuthenticated* state since the details the attacker sent match the details that the bank has on file.

Agreement

In UPPAAL I can also be more specific about the messages involved in the protocol.

For a strong agreement property, Security Property 5.1.5 specifies that after the branch has authenticated the customer with the old card, and the customer has verified the new card, then the bank and the customer must have the same details for the old card and the new card on file.

Security Property 5.1.5 (Card Agreement)

$$BankBrain.CustomerAuthenticated\,and\,CustomerBrain.VerifiedNewCard \dashrightarrow$$
$$BankBrain.receivedInfo == CustomerBrain.myCardDetails\ and$$
$$CustomerBrain.myCardDetails == newCardNumber$$

UPPAAL finds an attack on Security Property 5.1.5, pictured in Figure 5.15.

In this attack, the attacker impersonates a bank employee similarly to the attack in Figure 5.13 and delivers a dummy card that they have created to the customer at the billing address. However, to ensure that the customer has been authenticated in accordance with Security Property 5.1.5, the attacker leaves the billing address and goes back to the bank branch. There, the attacker authenticates the customer with the details that the customer sent to the attacker initially.

5.1.5 Conclusions

In modelling the New-Bank-Card Protocol, I have shown that the embeddings of the rules in my calculus into UPPAAL come into use when analysing a protocol beyond the already heavily studied and analysed NSPK protocol. This section has put into practice the techniques detailed in § 4.2 and found rather surprisingly complex attacks that are not immediately obvious. Although the New-Bank-Card Protocol is still not a "real" protocol (although it is *realistic*), it involves quite a few movements not only to execute but also to attack.

The traces of the attacks that UPPAAL outputs, upon finding that a security property is not satisfied, include the new locations of all of the mobile agents each time they move. This is a novel contribution to security protocol analysis, and my calculus and techniques of modelling will almost certainly come into use for analysis on other security protocol scenarios that feature movement of agents.

Keeping my arrangement in UPPAAL of a small number of models, i.e. one initiator, one responder, one attacker, one environment, still lends itself to surprisingly versatile scenarios to be instantiated for analysis. For instance, the moving part of the responder which is instantiated in the scenario in § 5.1.3 represents an employee. But any number of moving responders could be instantiated to represent any number of employees (and indeed customers on the initiator side) in order to analyse more complex properties. I do not show this explicitly here since the aim is to demonstrate an example of my approach in action by means of a case study, but this fact shows that there is a natural progression for the work to take. Also in keeping with the arrangement of initiator/responder/attacker/environment, this allows the New-Bank-Card protocol to be input into EnvGen for analysis on all possible environments straight away.

With UPPAAL on its own, one is still faced with the problem of analysing unbounded numbers of protocol sessions the way that other tools such as Tamarin and ProVerif can, since UPPAAL can only verify if one security property holds for one scenario of one protocol at a time. But these techniques could be developed further, with the expansion of not only EnvGen's technical

capabilities, but perhaps other applications which extend UPPAAL in other ways regarding scenario generation. With all this to be considered in future work, UPPAAL would make for a potentially very powerful security protocol analyser perhaps on par with the more bespoke security analysis tools.

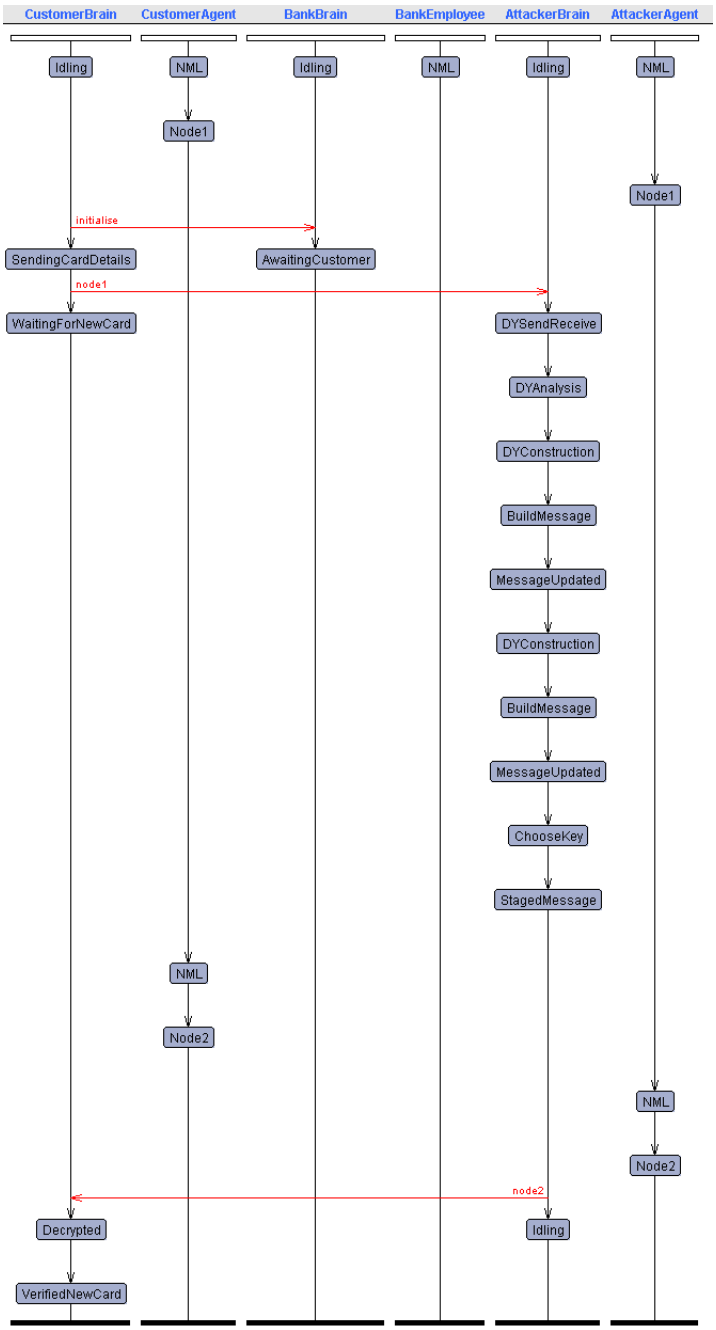

Figure 5.13: Attack on Security Property 5.1.1

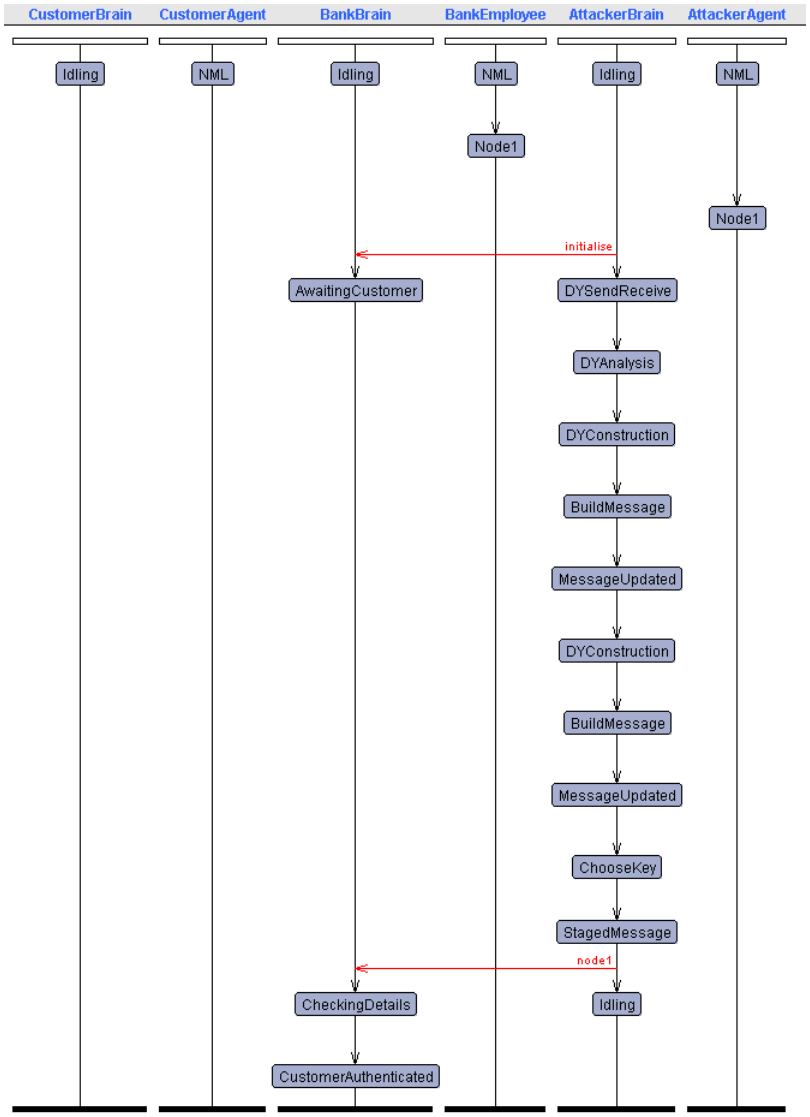

Figure 5.14: A Trace of a Possible Attack on Security Property 5.1.4

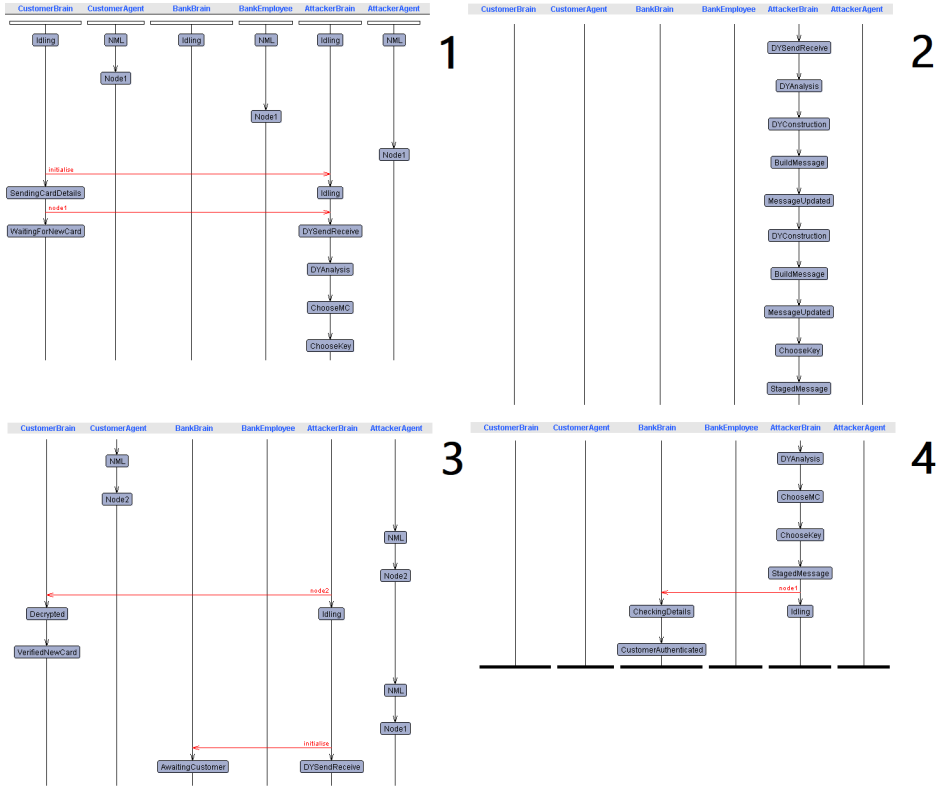

Figure 5.15: A Trace of a Possible Attack on Security Property 5.1.5

5.2 Case Study: The SegCom protocol

This final case study is the *SegCom protocol* [43], a three-phase security protocol between mobile vehicles and *road-side units (RSUs)*. The vehicles are mobile agents, whereas the RSUs are static agents that, critically, have a finite range of communication. Each vehicle and RSU possess a public-private key pair, allowing asymmetric encryption and signing, as well as certificates that they acquire beforehand from a trusted third party. Moreover, SegCom calls for ID-based encryption. In practice, the ID of an RSU, when used as part of ID-based cryptography, is its public key.

This type of network is a *Vehicular Ad-Hoc Network (VANET)*, and movement of the vehicles is a defining feature of the network. The highly dynamic nature of the network topology, the short time in which agents may be in range of communication, and the notion of trajectory in the movement make VANETs an interesting case study.

The SegCom protocol consists of three phases: (1) "Primary Authentication", (2) "Group formation for V2V Communication", (3) "Successive Authentication". I focus here on the primary phase of authentication between a mobile vehicle V_1 and an RSU_1 to establish a shared symmetric key, for further (more efficient) secure communication. First, V_1 must move in range of RSU_1 or no messages can be sent or received, and then:

- RSU_1 sends its certificate $Cert_{RSU_1}$ in clear-text upon V_1 entering the RSU's range.
- V_1 validates $Cert_{RSU_1}$ and then sends to RSU_1 the message $MV1 =$

$\{Cert_{V_1}, SK_{V_1,RSU_1}, \{SK_{V_1,RSU_1}\}_{SK_{V_1}}\}_{ID_{RSU_1}}$, where $Cert_{V_1}$ is V_1's certificate, SK_{V_1,RSU_1} is a freshly generated key shared by V_1 and RSU_1, $\{SK_{V_1,RSU_1}\}_{SK_{V_1}}$ is the same key signed by V_1 using its secret key, and ID_{RSU_1} is the public key of RSU_1.

- RSU_1 verifies the signature on the message using V_1's public key (obtained from the certificate, although for simplicity in my implementation the RSU will know the vehicle's public key already). RSU_1 can then be sure that the shared key SK_{V_1,RSU_1} has been created by V_1 and so can be used for subsequent secure communication.

- Finally, as V_1 is moving, eventually it leaves the communication range of RSU_1 and so the key SK_{V_1,RSU_1} is revoked by RSU_1, meaning that V_1 will need to re-certify itself with that RSU if it enters its range again.

Verma and Huang [43] assume that an attacker can deploy adversarial RSUs. In my approach, the attacker has much more power: it can play the role of a mobile hub or static node, depending on the scenario I consider. In this sense, in my analysis, vehicles and RSUs can be honest or dishonest.

5.2.1 The Protocol in Our Calculus

I first formalise the specification of SegCom's primary phase of authentication in the calculus from § 3. Figure 5.16 shows the steps in the case of $\mathcal{H} = \{\mathsf{V}_1\}$ and $Env = (\{RSU_1\}, \{\})$ and there is one node process RSU_1. In the analysis, I will consider more complex scenarios. I only show the trace of the vehicle

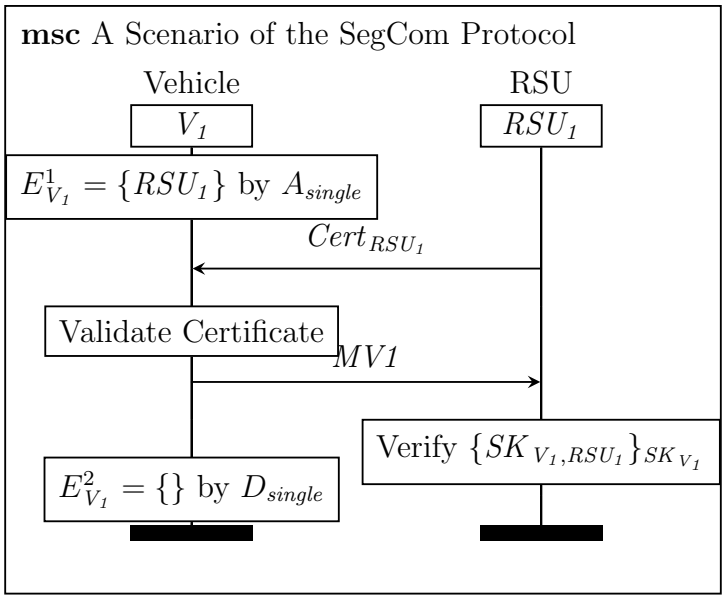

Figure 5.16: A Scenario of the SegCom Protocol

in my calculus, as the trace of the RSU is less interesting. As in § 5.1.1, the nodes have default channel names in accordance with the calculus.

$$
\begin{aligned}
\mathsf{V}_1 := \;& A_{single}(RSU_1).\\
& RSU_1(Cert_{RSU_1}).\\
& \overline{RSU_1}\langle\{Cert_{V_1}, SK_{V_1,RSU_1},\\
& \qquad \{SK_{V_1,RSU_1}\}_{SK_{V_1}}\}_{ID_{RSU_1}}\rangle.\\
& D_{single}(RSU_1).\ 0
\end{aligned}
$$

5.2.2 UPPAAL Implementation

Message Passing and Movement

In UPPAAL, for this case study, I allocate 5-bits for each message component. All message generation, analysis, and type-checking operations are done in the

```
int sign(int param1, int param2) {
      signatureindex+=1;
       if (signatureindex < signaturemax) {
         plain[signatureindex] = param1;
         key[signatureindex] = param2;
        return signatureindex;
       }
      return 0;
    }

int verify(int param1, int param2) {
      if (param1<signaturemax && key[param1]*10 == param2) {
       return true;
}
      return 0;
}
```

Figure 5.17: Signing and Verification Code for the SegCom Protocol in UPPAAL

same way as in § 4.3 and § 5.1. However, the SegCom protocol calls for the use of signing and verification, which were not used for the New-Bank-Card protocol. The functions that allow principals to carry these out can be seen in Figure 5.17.

The vehicle is the only moving agent in the protocol, and I label the state where the vehicle is not in range of any RSU as NML (no man's land) as before.

The Models

The models consist of one initiator, one responder, one attacker, and the environment. I then instantiate the models in UPPAAL as moving hubs or nodes which take on the role of a vehicle or RSU with respect to the scenario being

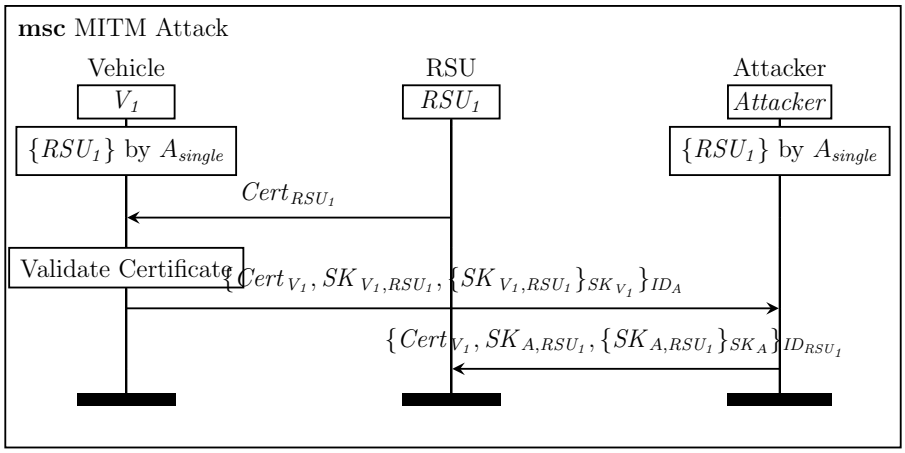

Figure 5.18: An Example MITM Attack on SegCom with Unsigned Certificates

considered.

Figure 5.19 shows the UPPAAL model of an initiator. The initiator sends its certificate to the responder, and waits to receive *MV1*. In the UPPAAL model, the initiator signs the initial certificate. This is due to the fact that in literature certificates are typically authentic and contain information regarding who sent the certificate itself. If certificates were to be sent in cleartext, the protocol could be trivially attacked by impersonation (see Figure 5.18). Upon receiving *MV1*, the initiator decrypts it and decomposes it into its constituent parts, storing them locally only if they type check and the signature is successfully verified with the responder's public key. If this is all successful, the initiator then officially certifies the responder.

Figure 5.20 shows the UPPAAL model for the responder. The responder first receives a message and checks that this is a valid and authentic RSU certificate. This transition is once again guarded by *boolParam* because the responder will have to move within range of an initiator in order to receive

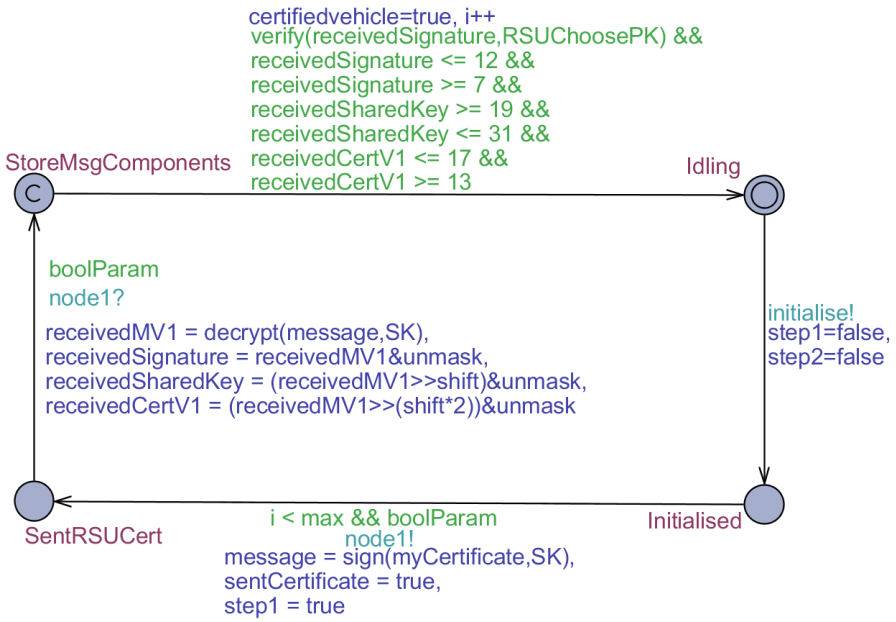

Figure 5.19: UPPAAL Model of the Initiator in the SegCom Protocol

this message. If it is a valid certificate, it sets a global variable *certifiedRSU* to true. The responder at this point is in the *BeginMV1Construction* state. The subsequent transition builds the message *MV1*. The responder generates a fresh shared key and signs it with its secret key. The key and the signed key are both concatenated onto the end of the certificate sent by the initiator, and this is then encrypted with the ID of the initiator. The responder then sends the encrypted *MV1* to the initiator on its default channel name.

To account for multiple scenarios that could take place with regards to who the principals choose to talk to (which public key they choose to encrypt messages), I provide separate scenario choice models akin to § 4.3. The RSU and the vehicle might choose to speak to the attacker, and as mentioned in § 3.6 the attacker can assume the role and capabilities of a node or hub at will.

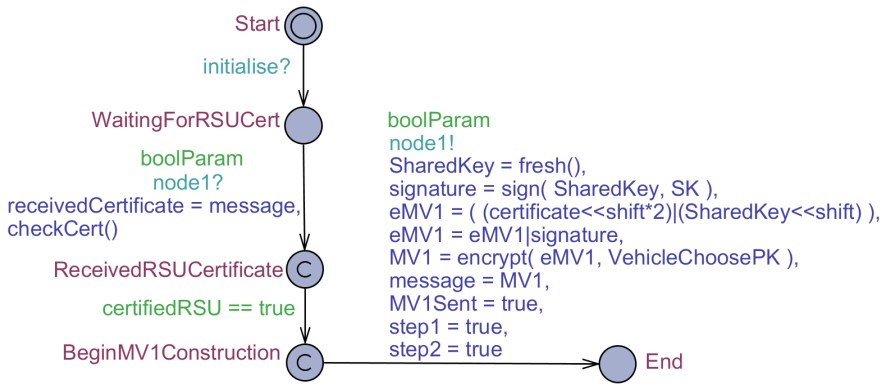

Figure 5.20: UPPAAL Model of the Responder in the SegCom Protocol

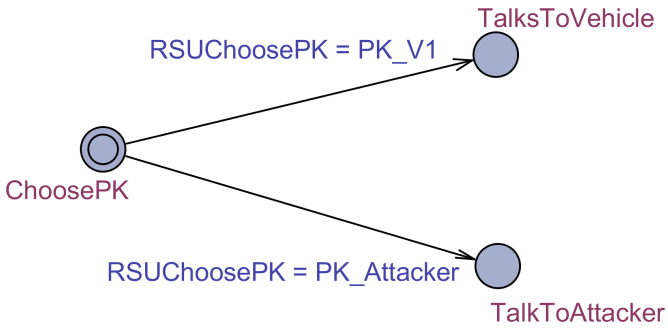

Figure 5.21: A Choice of Keys for The Initiator

As such, I account for the possible interactions among the principals. These choices are accounted for in Figures 5.21 and 5.22.

The Attacker and The Environment

The attacker obviously operates with the same capabilities as the attacker in § 5.1 and § 4.3. The only minor difference for this specific case study is that since the messages are longer than the case study in § 5.1, we give the attacker a larger number of maximum operations. Moreover, the attacker will now be able to sign and verify messages since SegCom uses asymmetric encryption.

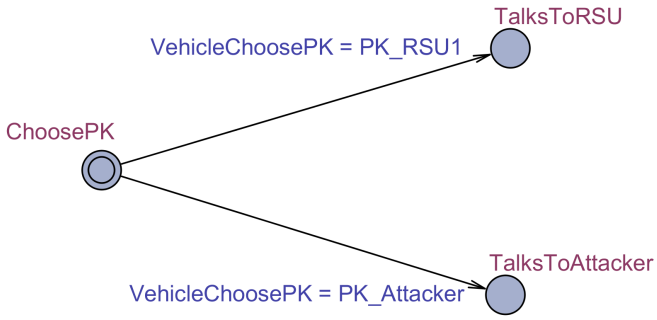

Figure 5.22: A Choice of Keys for The Responder

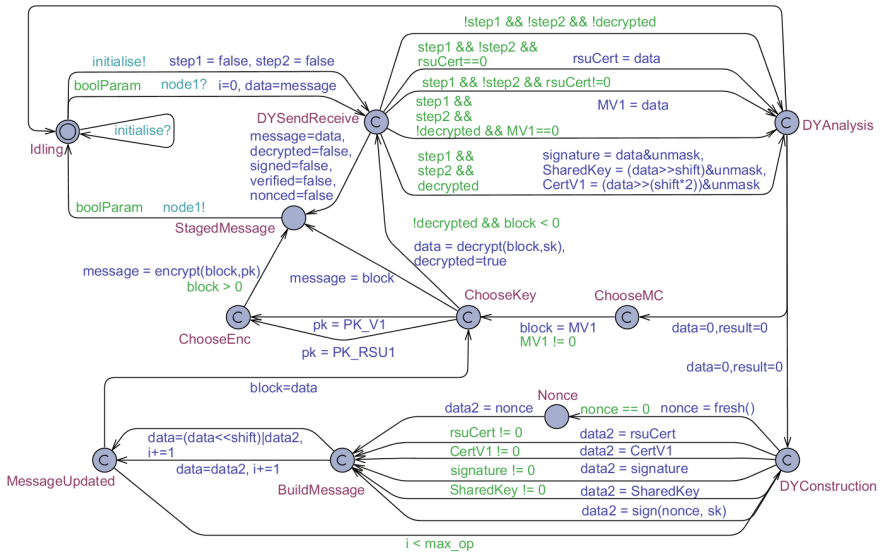

Figure 5.23: UPPAAL Model of the Attacker in the SegCom Protocol

I also assume for both scenarios, the attacker has been given a valid public-private key pair from a trusted third party. Figure 5.23 shows the attacker model in the case of the SegCom protocol.

Finally, I give the environment for this scenario in Figure 5.24. Crucially, a minor addition to this environment model is that when the vehicle moves out of the range of the RSU the key is no longer valid and the vehicle and RSU must carry out the protocol once more. This is justified in [43] by the authors

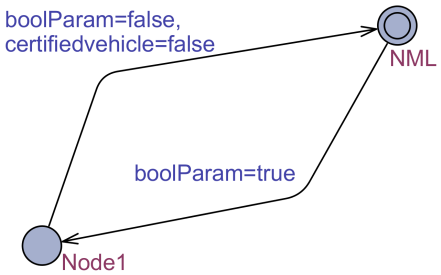

Figure 5.24: The Environment for the Scenario of the SegCom Protocol

mentioning that "the key is only valid for the segment (range of communication) for which it has been established."

5.2.3 Instantiating the Scenarios

As an example of my approach in action, I focus on a simple environment where $Env = (\{RSU_1\}, \{\})$. RSU_1 has an accompanying node processes RSU_1 and a single hub V_1 starts outside the communication range of RSU_1 and moves freely around the environment. Here, I have instantiated a single vehicle which can carry out an execution of the protocol with RSU_1 only. The reason is that I am looking for attacks on a single execution of the protocol, but the environment is still instantiated so that the tool can also search for attacks relating to movement and location. The RSU is the initiator of the protocol and the vehicle acts as the responder. The attacker I instantiate moves as a hub, and has no prior knowledge other than public keys of all the other principals. The attacker in instantiated in no man's land similar to the honest vehicle.

The instantiated scenario produces the Figures 5.26 through 5.32.

```
RSUScenarioChoice = SCRSU();
V1ScenarioChoice = SCVehicle();

VehicleAgent = Environment(vehicleInNode1);
AttackerAgent = Environment(inNode1);

VehicleBrain = Responder(vehicleInNode1,13);
AttackerBrain = Attacker(inNode1);
RSU1 = Initiator(isNode);
```

Figure 5.25: Instantiation of the Scenario for Analysis in UPPAAL

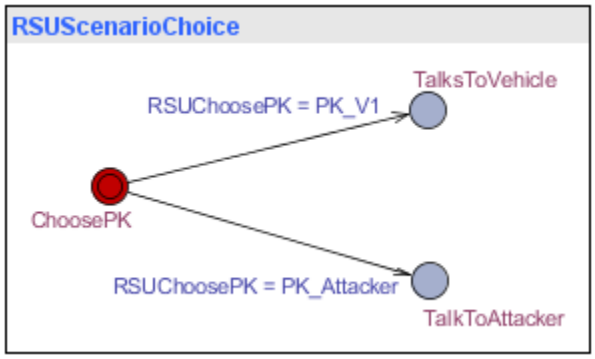

Figure 5.26: A Choice of Keys for the RSU

5.2.4 Analysis

As in § 5.1.4, I use the UPPAAL verifier with its requirement language to specify security properties, and check that they hold for a given scenario.

In the following, I report on the analysis of relevant security properties. According to the authors, this protocol aims to establish mutual authentication between a vehicle and an RSU, so as such I verify if security properties relating to authentication hold for the protocol.

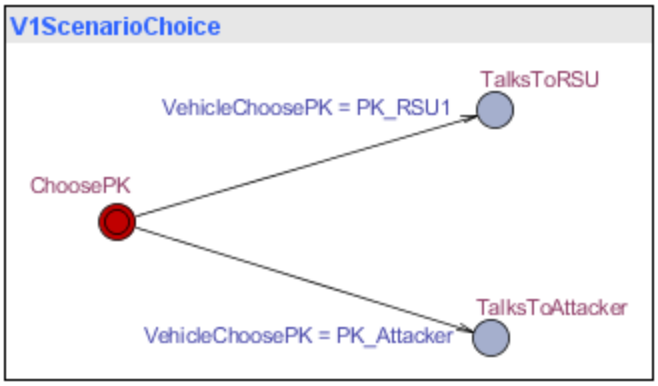

Figure 5.27: A Choice of Keys for the Vehicle

Aliveness

In this case, I define the vehicle as being alive to the RSU if, when the RSU has certified the vehicle, it is true that the vehicle entered the communication range of the RSU and certified the RSU, which is formalised as follows:

Security Property 5.2.1 (Aliveness of Vehicle)

$certifiedvehicle \dashrightarrow certifiedRSU$

For Security Property 5.2.1, the UPPAAL verifier finds Property 5.2.1 to hold.

However, consider the scenario where the attacker has obtained the certificate of a vehicle via a side-channel attack. For this scenario, UPPAAL finds an attack. The trace for this attack can be seen in Figure 5.33. This attack is unsurprising, since the attacker knows the certificate of an honest vehicle already. The RSU simply chooses to talk to the attacker when they move within range, and the attacker can simply use this certificate in the *MV1* message that they create and send to the RSU.

As an aliveness property of the RSU, I define in Security Property 5.2.2

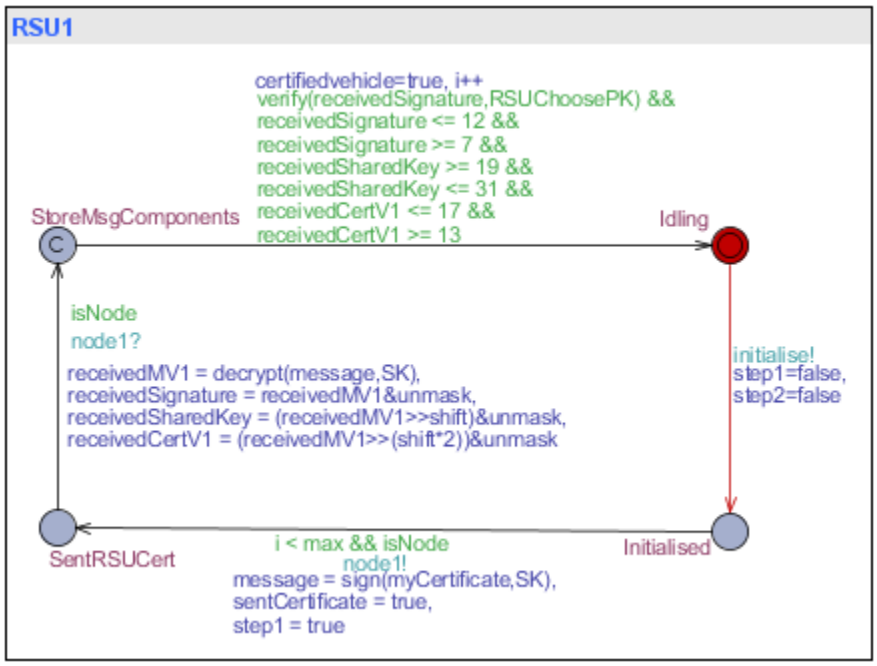

Figure 5.28: The Instantiated RSU

that the RSU is alive to a vehicle if, when the vehicle sends message *MV1*, the RSU had previously sent its certificate.

Security Property 5.2.2 (Aliveness of RSU)

VehicleBrain.End $\dashrightarrow$ *RSU1.sentCertificate*

Security Property 5.2.2 is found to hold for this scenario. However, unlike for Security Property 5.2.1, a separate scenario in which the attacker has been able to obtain the certificate of the RSU_1 via some side channel attack does not yield an attack either. This is due to the fact that the certificates sent by RSU are signed.

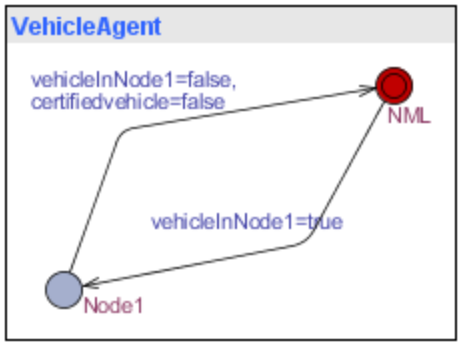

Figure 5.29: The Instantiation of the Movement Part of the Vehicle

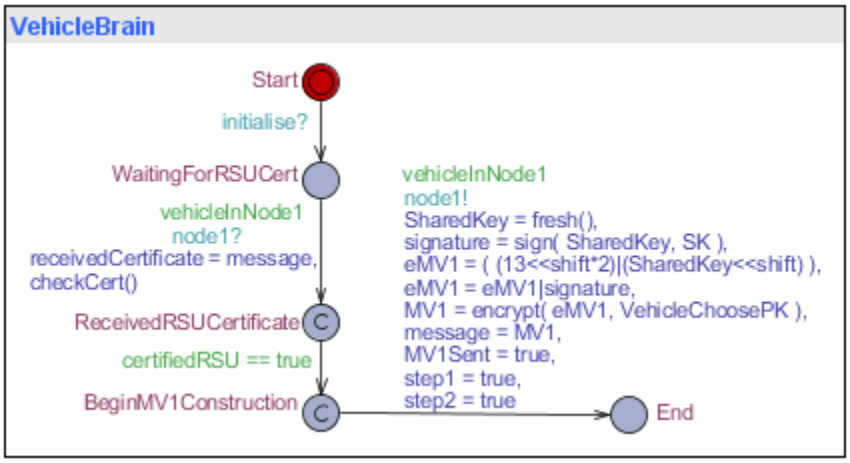

Figure 5.30: The Instantiation of the Message Passing Part of the Vehicle

Shared Key Agreement

The vehicle and RSU_1 should agree on the shared key that is established in a single protocol execution, immediately after $MV1$ is received, decrypted, and the signature verified by RSU_1.

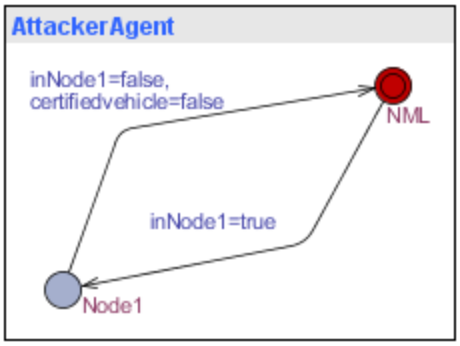

Figure 5.31: The Instantiation of the Movement Part of the Attacker

Security Property 5.2.3 (Shared Key Agreement)

$$certifiedvehicle \text{ and } RSU1.Idling \text{ and } certifiedRSU \text{ and } VehicleBrain.End \dashrightarrow VehicleBrain.SharedKey = RSU1.receivedSharedKey$$

UPPAAL finds this strong key agreement to hold. Figure 5.18 however, shows an attack that could be possible if the initial certificate sent by the RSU was not signed.

Location Verification

Finally, if RSU_1 certifies the vehicle, then the vehicle must be situated in range of RSU_1.

Security Property 5.2.4 (Location Verification)

$certifiedvehicle \text{ and } certifiedRSU \dashrightarrow VehicleAgent.Node1$

For Security Property 5.2.4, UPPAAL finds an attack. In the attack, the attacker intercepts the message *MV1* and holds this message until the vehicle

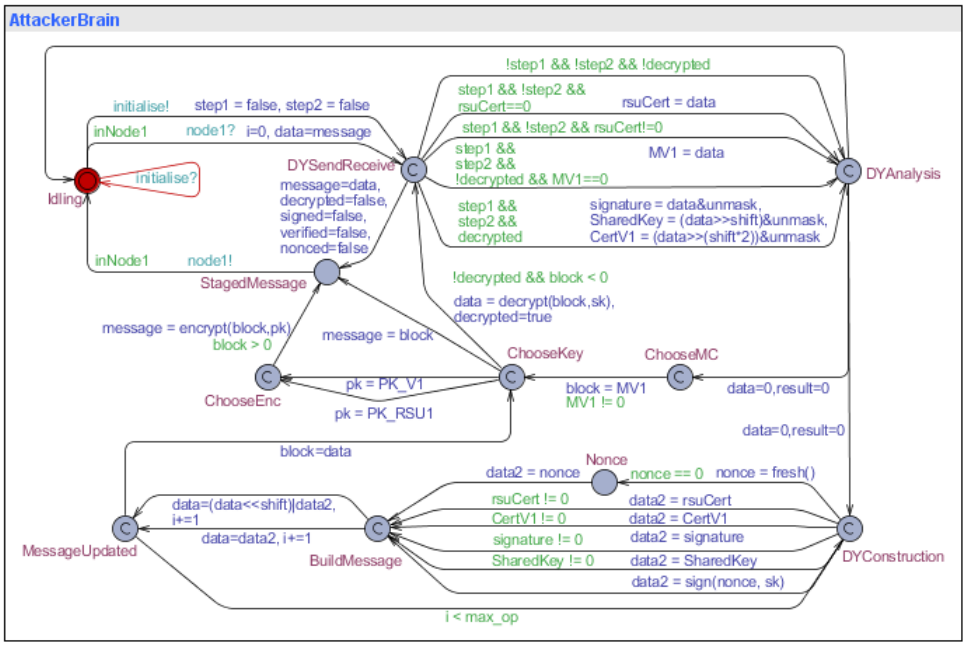

Figure 5.32: The Instantiation of the Message Passing Part of the Attacker

has left the communication range of the RSU. The attacker then finally sends the message *MV1* to the RSU so that the RSU certifies a vehicle that is now far away. It is thus too late to have any subsequent communication with this vehicle. This attack can be seen in Figure 5.34.

5.2.5 Conclusions

Much like the New-Bank-Card Protocol case study, this case study has put into practice the techniques detailed in § 4.2 and found rather surprisingly complex attacks that are not immediately obvious. Even though the environment itself is perhaps the most basic, there are still some attacks to be found which involve movement and the locations of principals at certain times. This chapter also shows how the implementation of a security protocol and the instantiation of

a scenario is relatively consistent across different case studies. The techniques are the same, the *general* attacker model is the same (albeit working with different data for each case study) and the method to instantiate different scenarios is the same.

As mentioned in § 5.2, SegCom consists of more than just one protocol; it has three phases. This section focuses on finding potential attacks with movement for the "Primary Authentication" phase but I suggest future work to involve case studies for the other two phases, "Group formation for V2V Communication" and "Successive Authentication". It would be ideal to be able to compose these protocols together for a full analysis of the three phases together. This would allow for a much stronger verification with regards to the security properties that would be analysed. This could also allow for EnvGen to be utilised, since more locations would be involved. Currently there is only one RSU in the environment considered in this case study, therefore there is only one possible environment. But with multiple RSUs performing successive authentications and multiple vehicles involved in the scenario, the results become stronger and the benefits of EnvGen become more apparent.

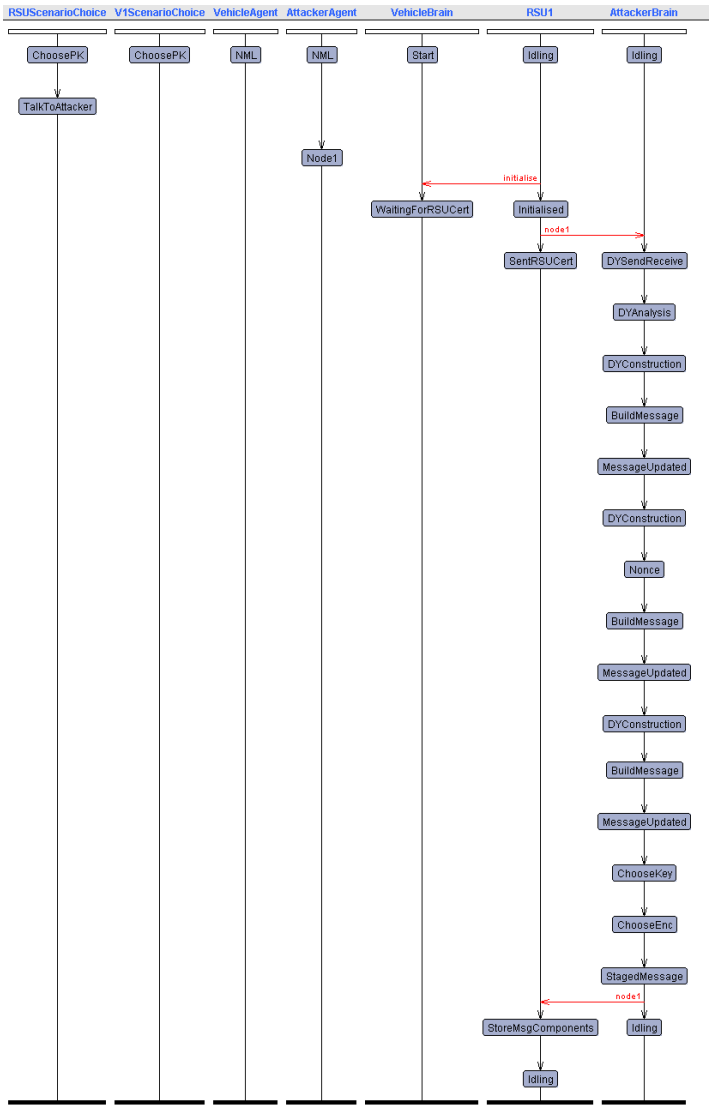

Figure 5.33: Trace of a Possible Attack on Security Property 5.2.1

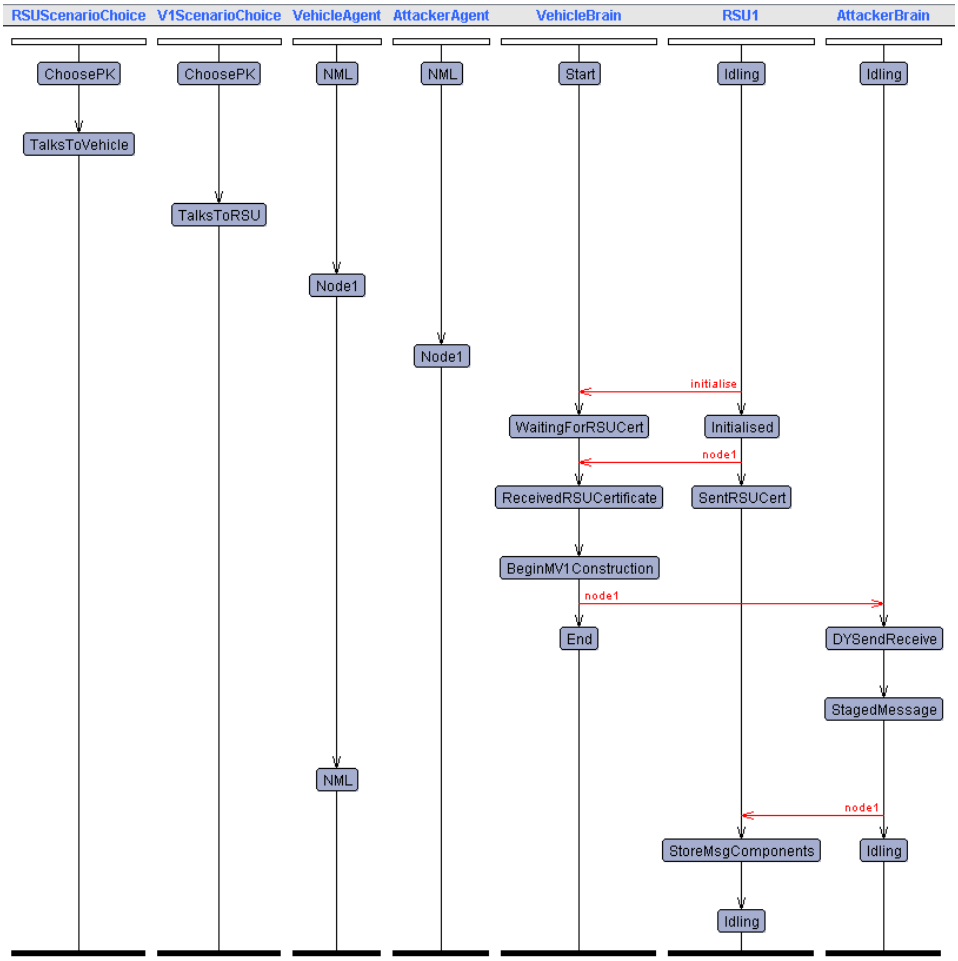

Figure 5.34: Attack on Security Property 5.2.4

Chapter 6

Conclusions and Future Work

This final chapter gives concluding remarks on the work presented in this thesis and on future work.

6.1 Conclusions

My calculus allows me to consider explicitly security protocols that require the principals to move to specific locations in order to exchange particular messages, along with a Dolev-Yao attacker who is able to move at will. With UPPAAL's support, I have automatically analysed some properties of the two case studies that I considered, discovering attacks that rely explicitly on movement. The extended pi-calculus that I created showed how a standard security protocol trace can be enhanced to annotate the movement of the involved principals. The two case studies on the New-Bank-Card protocol and the primary authentication phase of SegCom show how the embedding of the calculus rules in the tool as well as the tool itself can be used as a solid foundation for analysis on security protocols with movement.

As more and more of the technology we use in our daily life becomes mobile, it is going to be increasingly more important to factor into security protocol analyses the locations of the protocol participants and how they move. But movement as a factor in a security protocol is not limited to just mobile hardware. Movement can already be observed in security protocols without specialist hardware such as the New-Bank-Card protocol which only involves the movement of humans.

This thesis, as I have said in § 3.7, does not propose a solution to capturing

any and all types of movement. The movement I am capturing relates specifically to location and does not take into account other physical factors at play such as height, velocity, and so on. These factors would be quite difficult to implement in UPPAAL alone, and I do not model *exact* locations of hubs with real-valued co-ordinates or otherwise, so this would need to be done before a tool is considered. Likewise, messages are assumed to be sent and received in negligible time, but one might want to implement delays in message sending and receiving, of which work has been done already in [26].

Perhaps the largest hurdle when approaching this problem was the fact that most protocols which involved physically disperse principals used routing to establish long distance connections and forwent the idea of moving near one another altogether. Distance-bounding protocols of course explicitly deal with location verification, but do not talk about *how* a principal arrives at a location in the first place.

In conclusion, I believe that my thesis contributes to the broader study of security of mobile agents by looking specifically at how movement and location play a role in the verification of security properties. I have been able to model this formally and analyse security protocols in this domain automatically by means of a tool, chosen specifically for its advantages with regards to capturing the types of movements I consider.

6.2 Future Work

This section makes suggestions on future research to be done in this area.

For the extended applied pi-calculus, there are a few properties that might be beneficial to prove. The most immediate next step for the calculus would be to prove completeness of the movement semantics, i.e. prove that the rules cover all possible movements. A denotational semantics to accompany the operational semantics that I have already provided for movement and message passing would also be beneficial.

An interesting research question to explore might be to prove certain properties that relate my extended mobile Dolev-Yao intruder model with the standard model. For instance, it is quite obvious that for every attack in the standard Dolev-Yao model there exists an attack in the model with movement. Similarly, it is quite straightforward to show that a dishonest node with infinite communication range in my model is equivalent to the standard Dolev-Yao attacker scenario. What will be more challenging is to identify if and when attacks in my model, in which the dishonest node does not have infinite range, can be reduced to attacks in the standard model, or to show that there are attacks on certain protocols that must require movement, where attacks in the standard Dolev-Yao model do not exist.

"Escaping" movements are one of many potential ways that a hub could thwart an attacker. In other words, if in NSPK with movement Alice decides to abort the protocol by staying in no-man's-land, the attack is no longer executable. The extent to which the hub movements can prevent attacks is a question I would like to explore in more detail.

As mentioned in § 5.1.5 there is a drawback in UPPAAL of not being able to analyse unbounded numbers of protocol sessions the way that other tools

such as Tamarin or ProVerif can. I suggest future work could be to amend this drawback (or at least mitigate it) by finding a way to structure the knowledge of the agents better, particularly of the attacker. At the moment they are simply stored in the models of initiators, responders, and attackers as individually defined local variables which must be changed by hand for each scenario, but an easier way to instantiate scenarios with different knowledge is preferable. EnvGen follows naturally from UPPAAL because the templates that represent each machine are stored locally as XML files which are highly structured, but being able to express knowledge of the agents in a highly structured way such as this would expand the number of scenarios that could be considered even more. The problem here, of course, could be that UPPAAL runs out of memory during a search for attacks. High-performance computing power might be required here.

With regards to what has already been done with EnvGen, there is still plenty more to be done. There is more functionality still to be included, such as manipulating environments from within EnvGen (adding and removing nodes for instance), improving the console output (mentioned in 4.5), and testing the application thoroughly and carefully.

All this considered, the above future works should be complemented with even more complex case studies beyond what has been considered here already. For instance, a principal might need to move in a certain way to avoid their movements being detected by an attacker, should this be the goal of the security protocol. Other more complex security protocols might be multi-phase and require movement or feature unusual or more subtle movements.

Bibliography

[1] Dick cheney: Heart implant attack was credible. *BBC. (2013)*, Retrieved from: http://www.bbc.co.uk/news/technology-24608435.

[2] E. A. Akkoyunlu, K. Ekanadham, and R. V. Huber. Some constraints and tradeoffs in the design of network communications. In *Proceedings of the fifth ACM symposium on Operating systems principles*, pages 67–74, 1975.

[3] R. Baheti and H. Gill. Cyber-physical systems. *The impact of control technology*, 12(1):161–166, 2011.

[4] F. Bai, N. Sadagopan, and A. Helmy. The IMPORTANT framework for analyzing the Impact of Mobility on Performance Of RouTing protocols for Adhoc NeTworks. *Ad hoc networks*, 1(4):383–403, 2003.

[5] D. Basin. *The Cyber Security Body of Knowledge*, chapter Formal Methods for Security. University of Bristol, 2021. Version 1.0.

[6] D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski. Arpki: Attack resilient public-key infrastructure. In *Proceedings*

of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pages 382–393, 2014.

[7] D. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse, and V. Stettler. A formal analysis of 5G authentication. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 1383–1396, 2018.

[8] M. Baudet. Random polynomial-time attacks and dolev-yao models. *J. Autom. Lang. Comb.*, 11(1):7–21, 2006.

[9] B. Blanchet. Automatic verification of security protocols in the symbolic model: The verifier proverif. In *Foundations of security analysis and design VII*, pages 54–87. Springer, 2013.

[10] S. Brands and D. Chaum. Distance-Bounding Protocols. In *Workshop on the Theory and Application of of Cryptographic Techniques*, LNCS 765, pages 344–359. Springer, 1993.

[11] M. Bugliesi, G. Castagna, and S. Crafa. Reasoning about security in mobile ambients. In *Proceedings of CONCUR*, LNCS 2154, pages 102–120. Springer, 2001.

[12] L. Cardelli and A. Gordon. Mobile ambients. *Theoretical computer science*, 240(1):177–213, 2000.

[13] I. Cervesato. The dolev-yao intruder is the most powerful attacker. In *16th Annual Symposium on Logic in Computer Science—LICS*, volume 1, pages 1–2. Citeseer, 2001.

[14] L. Chen and M. Ryan. Attack, solution and verification for shared authorisation data in tcg tpm. In *International Workshop on Formal Aspects in Security and Trust*, pages 201–216. Springer, 2009.

[15] T. Chothia and M. D. Ryan. Modelling of 802.11 4-Way Handshake Attacks and Analysis of Security Properties. In *STM 2020: Security and Trust Management*, LNCS 12386, pages 3–22. Springer, 2020.

[16] A. Cook and L. Viganò. A Game Of Drones: Extending the Dolev-Yao Attacker Model With Movement. In *Proceedings of the 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 280–292. IEEE, 2020.

[17] J. Cordasco and S. Wetzel. An attacker model for MANET routing security. In *2nd ACM conference on Wireless network security*, pages 87–94. ACM, 2009.

[18] R. Corin, S. Etalle, P. H. Hartel, and A. Mader. Timed analysis of security protocols. *Journal of Computer Security*, 15(6):619–645, 2007.

[19] C. Cremers, K. B. Rasmussen, B. Schmidt, and S. Capkun. Distance Hijacking Attacks on Distance Bounding Protocols. In *IEEE Symposium on Security and Privacy*, pages 113–127. IEEE, 2012.

[20] J. Daemen and V. Rijmen. Reijndael: The advanced encryption standard. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, 26(3):137–139, 2001.

[21] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[22] E. K. K. Edris, M. Aiash, and J. Loo. Formal verification of authentication and service authorization protocols in 5g-enabled device-to-device communications using proverif. *Electronics*, 10(13):1608, 2021.

[23] A. E. Fallah-Seghrouchni and A. Suna. Claim: A computational language for autonomous, intelligent and mobile agents. In *International Workshop on Programming Multi-Agent Systems*, pages 90–110. Springer, 2003.

[24] J. P. Farwell and R. Rohozinski. Stuxnet and the future of cyber war. *Survival*, 53(1):23–40, 2011.

[25] A. Hinds, M. Ngulube, S. Zhu, and H. Al-Aqrabi. A Review of Routing Protocols for Mobile Ad-Hoc NETworks (MANET). *International journal of information and education technology*, 3(1):1, 2013.

[26] A. Koltuksuz, B. Kulahcioglu, and M. Ozkan. Utilization of timed automata as a verification tool for security protocols. In *2010 Fourth International Conference on Secure Software Integration and Reliability Improvement Companion*, pages 86–93. IEEE, 2010.

[27] C. Kuratowski. Sur le probleme des courbes gauches en topologie. *Fundamenta mathematicae*, 15(1):271–283, 1930.

[28] R. Küsters and T. Truderung. Using proverif to analyze protocols with diffie-hellman exponentiation. In *2009 22nd IEEE Computer Security Foundations Symposium*, pages 157–171. IEEE, 2009.

[29] R. Lanotte, M. Merro, A. Munteanu, and L. Viganò. A formal approach to physics-based attacks in cyber-physical systems (extended version). *arXiv preprint arXiv:1902.04572*, 2019.

[30] G. Lowe. An attack on the needham- schroeder public- key authentication protocol. *Information processing letters*, 56(3), 1995.

[31] S. Lu, L. Li, K.-Y. Lam, and L. Jia. SAODV: a MANET routing protocol that can withstand black hole attack. In *2009 International Conference on Computational Intelligence and Security*, volume 2, pages 421–425. IEEE, 2009.

[32] J. E. Martina, E. Dos Santos, M. C. Carlos, G. Price, and R. F. Custódio. An adaptive threat model for security ceremonies. *International Journal of Information Security*, 14(2):103–121, 2015.

[33] S. Mauw, Z. Smith, J. Toro-Pozo, and R. Trujillo-Rasua. Distance-Bounding Protocols: Verification without Time and Location. In *IEEE Symposium on Security and Privacy*, pages 549–566. IEEE, 2018.

[34] C. A. Meadows. Analyzing the needham-schroeder public key protocol: A comparison of two approaches. In *European symposium on research in computer security*, pages 351–364. Springer, 1996.

[35] S. Meier, B. Schmidt, C. Cremers, and D. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *Proceedings of CAV*, LNCS 8044, pages 696–701. Springer, 2013.

[36] G. E. Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998.

[37] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[38] A. Omicini and F. Zambonelli. Coordination for internet application development. *Autonomous Agents and Multi-agent systems*, 2(3):251–269, 1999.

[39] R. L. Rivest. Cryptology. 1990.

[40] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public key cryptosystems. In *Secure communications and asymmetric cryptosystems*, pages 217–239. Routledge, 2019.

[41] D. Singelee and B. Preneel. Location verification using secure distance bounding protocols. In *IEEE International Conference on Mobile Adhoc and Sensor Systems Conference*. IEEE, 2005.

[42] M. Uma and G. Padmavathi. A survey on various cyber attacks and their classification. *Int. J. Netw. Secur.*, 15(5):390–396, 2013.

[43] M. Verma and D. Huang. SeGCom: secure group communication in VANETs. In *Proceedings of the 6th IEEE Consumer Communications and Networking Conference*, pages 1–5. IEEE, 2009.